

編集

Category

DELETE

書式

DELETE arg1 [arg2]

使い方

DELETE n
DELETE n m

機能

指定行の抹消

解説

行あるいは範囲を指定してプログラムを抹消します。

ERASE

書式

ERASE

使い方

機能

FLASH ROMの消去

解説

FLASH ROMの消去。システム入れ替え時は、MPCINIT後ERASEしてください。

FREE

書式

FREE

使い方

機能

残容量の表示

解説

残容量をバイト数で表示します。

```
#free
176500
```

LIST

書式

LIST arg1 [arg2]

使い方

LIST 10 3
LIST *AHO
LIST

機能

プログラムリストを表示する。

解説

1番目の引数は、表示指定文番号です。2番目の引数は、表示行数です。
LIST単独の場合は、先頭から表示します。無引数のままLISTを実行すると続きを表示します。

MPCINIT

書式

MPCINIT

使い方

機能

MPCを初期状態にする。

解説

プログラムエリアを消去、変数・点データ・配列エリアを0にします。
I/OエリアをすべてOFF(クリア)状態とします。

NEW

書式

NEW

使い方

機能

プログラム消去

解説

プログラムを消去し予約変数以外の変数を抹消します。

RENUM

書式

RENUM [n]

使い方

RENUM
RENUM 5

機能

文番号の振り替え

解説

文番号を10ごとに振りなおします。数を指定するとその数で振りなおします。

TAIL

書式

TAIL

使い方

機能

最大文番号の表示

解説

最大文番号の表示。オンラインでプログラムを追加する時に使います。

保守

Category

BAT

書式

BAT(arg)

使い方

```
IF BAT(0)==1 THEN : PRINT "Battery error" : END_IF
```

機能

バッテリー エラー番号を得る

解説

前回の電源オフ時に正しくCPUが退避状態になったかどうかを示す関数です。0で正常。
1が返ってきたら、電源断時のCPU異常、又はバックアップ電池が尽きている。のどちらかです。
バッテリー・エラーがあった場合、点データ、MBKデータなどが破壊されている可能性があります。

BREAK_POINT {BKP}

書式

BKP [args]

使い方

BKP 100
BKP 100 110
BKP *aa
BKP 0

機能

ブレークポイントの設定

解説

BREAK_POINTコマンドにより、8個までの指定した文番号でプログラムを停止させることができます。(ラベル指定も可能です)

サンプルプログラムのようにプログラム番号を指定すると、指定行を表示します。

その後指定行の文番号は、FTMW上では反転表示されます。

ブレークポイントは、順々に文番号を指定します。指定した文番号を解除する場合は、同じ番号を入力します。

どの文番号が登録されているかは、BKPコマンドを引数なしで実行します。

また、すべてのブレークポイントを解除するには、BKP 0と入力します。

ブレークが発生したら

実際にブレークポイントを指定してRUNさせると、指定位置で実行が中断されます。

そして、中断した行と、タスク番号が表示されます。nキーによって、次のブレークポイントまで実行再開します。

このプログラムでは、文番号30を通るたびに実行前にブレークします。

ステップ送り(一行ずつ継続的に実行)させる場合はtを押します。ステップ送りの解除は、nを押します。

ブレーク停止中に変数や関数の値を参照することができます。

‘ p ’ を押して、続けて変数名や関数名を入力します。

ブレークポイントを追加することもできます。

‘ b ’ を押して文番号を入力すると、ブレークポイントを追加することができます。

ブレーク中にそのブレークポイントを解除したい場合は、” u ” を入力します。

プログラム実行を停止する場合は ‘ e ’ を押します。

なお、FTMW6.39s以後、ブレークポイントはメニューで使用できます。

```
30 FORK 2 *bb
40 END
110 *bb
120 DO
130 FOR i_=8 TO 15
140 ON i_: TIME 50: OFF i_
150 NEXT
160 LOOP
#bkp 110 140

110 *bb
140 ON i_: TIME 50: OFF i_
#bkp
BREAK_POINT 0=110
BREAK_POINT 1=140
#bkp 110

110 *bb
#bkp
BREAK_POINT 0=140
#
```

CTRL_A

書式

CTRL_A [val]

使い方

CTRL_A 1
CTRL_A 0

機能

CTRL_A機能設定

解説

プログラムポートでSOH(CTRL_A)を受信したあとのプログラム起動の可否を指定します。

CTRL_A 0 : J1-5,6状態がオープンであれば、プログラムを再起動します。(標準状態)

CTRL_A 1 : J1-5,6の状態にかかわらず、プログラムの再起動は行いません。

ENG

書式

ENG

使い方

機能

英語モードにする。

解説

MPCINIT後は英語モードとなっている。エラー表示は英文となります。

INSPEC

書式

INSPEC

使い方

機能

セルフテスト

解説

現在はRAMのWrite/Readテストのみです。

```
#inspec
INSPECTION
1:Test Memory
PASSED
#
```


JPN

書式

JPN

使い方

機能

日本語モードにする。

解説

日本語モードにします。エラー表示は日本語となります。MPCINIT後は英語モードとなっています。

LABELS

書式

LABELS

使い方

機能

ラベル検査

解説

ラベルの二重定義を検査します。二重ラベルを発見すると以下のように表示します。

The two same labels

12810 *bb

13400 *bb

LOCK

書式

LOCK LockKey
LOCK -1 LockKey

LockKey は正数 (2147483647以下)

使い方

LOCK 1234567
LOCK -1 1234567

機能

プログラムの秘密化、変更禁止

解説

プログラムを秘密化します。

LOCKコマンドはRUN 前、RUN後いずれにも使用できます。

RUN前にLOCK した場合は、必ずRUNを実行します。

解除は、最初の引数を-1 として、設定時と同じ値をいれます。

LOCKが解除され、再度編集可能な状態となります。

そのまま再ロックする場合は、ERASEを実行してからLOCKを実行します。

継続編集を必要としない場合は、MPCINIT,ERASE でも、LOCKは解除されます。

LOG

書式

LOG [arg]

使い方

LOG
LOG 0
LOG 1

機能

ログ表示

解説

LOGは実行中にプログラムポートに出力された文字の記録です。

LOGバッファNEW,もしくは、LOG 0でクリアされます。

停止時あるいは稼動中にLOGコマンドによってプログラムポートの出力を表示します。

20行ごとに順々に表示しますので、LOGコマンドを繰り返します。

最初から表示する場合は、LOG 1とします。LOGを初期化するには、LOG 0を実行します。

LOGコマンドを実行するとLOGは停止します。再開させるためにはLOG 3を実行します。

稼動中の装置でLOGを実行をして様子をみたあとはかならずLOG 3を実行してLOGを継続させます。

MON

書式

MON [arg]

使い方

MON
MON 1
MON 2

機能

実行状態の確認 監視

解説

プログラム停止後、monを実行すると以下のような表示されます。

```
#mon
```

```
*0_ [20] *1 [42] *2q [42] *3! [42]
```

```
#
```

0_ は、タスク0がENDで終了したことを意味しています。

2q は、タスク2がQUITコマンドで終了させられたことを意味しています。

3! は、タスク3が実行中で時間ロスがあることを示しています。

以下はタスク0を停止状態(コマンド受付可能) にしてタスクの状態を監視した様子です。

他のタスクからQUITされると、文番号が残ってQUITとなり、ENDで停止すると4_のようにタスク番号に_が付与されます。

SLPIは、PAUSEされたかTIMEコマンドによる停止中を意味します。

```
#mon 1
```

*1 RUN [42] *2 QUIT[42] *3 SLP [42] *4_ QUIT[120]

*5! RUN [140]

MON 2 を実行すると、表示はされずLOGデータに書き込みます。

通常は、コマンドとして使用しないで、プログラム中に埋め込み、

プログラム実行中に特定の箇所で、全体の実行状態を確かめるために有効な機能です。

PRA

書式

PRA array(n)
PRA var_[n]

使い方

PRA J_
PRA J_n
PRA AHO(10)
PRA FOOL(10,1)

機能

配列値の表示

タスクローカル変数の表示

解説

タスクローカル変数を表示します。
第二引数で、タスク番号を指定できます。
指定されない場合は、すべてのタスクの変数値を表示します。

配列の中身をまとめて表示します。
配列要素を20個ずつ表示します。
二次元配列にも対応します。

PRINT

書式

PRINT [val,str]

使い方

```
PRINT "res=" a$ a cc bb$ a a a$ "123abc"
```

機能

数値 文字列の表示 デバッグ用

解説

変数や文字列を表示します。

プログラム中に入れて状態監視に使用します。

```
10 a$="123"
15 bb$="koatae"
20 a=456 : cc=1096
30 PRINT "res=" a$ a cc bb$ a a a$ "123abc"
#run

res= 123 456 1096 koatae 456 456 123 123abc
#
```


PRX

書式

PRX val

使い方

PRX A

機能

ヘキサ形式表示

解説

数値をヘキサ形式で表示します。

デバッグ用コマンドです。

プログラム中で文字列としてのHEX表現が必要な場合はHEX\$()を用います。

```
A=100:B=1000:C=10000
prx A B C
00000064 000003E8 00002710
#
```

RS

書式

RS ch

使い方

RS 1

機能

受信バッファ表示

解説

コマンドを実行すると、以下のように256byte受信バッファを表示します。

末尾が最も最近受け取った文字となります。

">"が現在の未読文字位置をしめています。INPUTではここから読み始めます。

#rs 1

The '>' shows the 1st chr-byte

```
5 -CR [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
[D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
[D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
[D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
[D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
[D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
[D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
[D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
>1 1 1 1 3 3 3 n m n m n m CR ~
```

#

SLOW_RUN

書式

```
SLOW_RUN taskn [ timer ]  
SLOW_RUN TMOUT [n]
```

使い方

```
SLOW_RUN 1 100  
SLOW_RUN TMOUT 1000
```

機能

指定タスクの遅延実行
ダウンカウンタ時間設定

解説

SLOW_RUNは、引数により、以下の二通りの使い方があります。

例1) SLOW_RUN 1 1000

この場合は、タスク1の実行ステップごとに1000msec時間待ちを指定することになります。

この値はプログラム実行中にも変更できます。

デバッグの最初はこのコマンドにより、慎重にゆっくりプログラムを実行させ、

デバッグが進展するにしたがってより早く実行させるようにします。

プログラムの安全が確認できたら、"SLOW_RUN 1"を実行します。

このように、引数にタスク番号のみを指定すれば、タイマー待ちは解除されます。

例2) SLOW_RUN TMOUT 1000

引数に予約定数"TMOUT"を追加すると、タイムアウトダウンカウンタの設定となります。

通常、ダウンカウンタは通常100m秒ごとにダウンカウントしますが、

この例のように100以上の値を設定すると、指定msec値ごとのダウンカウントとなります。

この例では、1000msec(1秒)ごとにダウンカウントとなります。

なお、SLOW_RUNでの設定は、パワーオンリセットで解除されますが、

プログラム中に記述すると遅延要素が不用意に設定されてしまうのでコマンドとして使用してください。

STACKS

書式

STACKS

使い方

機能

スタックエリアの消費状態を表示する。

解説

STACK FREEは未使用のスタックエリアのロング・ワード数。

POSは、現在のスタックポインタの位置です。ロング・ワードのカウンタ数で表示されます。0となっているのは、まだ起動されていないタスクです。

```
#stacks
TASK0 STACK FREE=156 STACK POS =38
TASK1 STACK FREE=200 STACK POS =0
TASK2 STACK FREE=200 STACK POS =0
TASK3 STACK FREE=200 STACK POS =0
TASK4 STACK FREE=200 STACK POS =0
TASK5 STACK FREE=200 STACK POS =0
TASK6 STACK FREE=200 STACK POS =0
TASK7 STACK FREE=200 STACK POS =0
TASK8 STACK FREE=200 STACK POS =0
TASK9 STACK FREE=200 STACK POS =0
TASK10 STACK FREE=200 STACK POS =0
TASK11 STACK FREE=200 STACK POS =0
TASK12 STACK FREE=200 STACK POS =0
TASK13 STACK FREE=200 STACK POS =0
TASK14 STACK FREE=200 STACK POS =0
TASK15 STACK FREE=200 STACK POS =0
#
```

VARs

書式

VARs [arg]

使い方

VARs
VARs VOID
VARs 0

機能

変数のリスト

解説

1) 引数無し

4文字以上、1文字のみ相違のある変数を表示します。紛らわしい変数を表示することによって変数による混乱を減らします。

2) VOID

RUN後に実行します。"VARs VOID"

実行中に一度も値が代入されない変数を表示します。これは初期化が不完全であったり、誤って使われたラベルの発見に便利です。

3) 値 "VARs 0"

値を指定すると、その値を持つ変数を表示します。

VER

書式

VER

使い方

#VER

機能

バージョン表示

解説

MPC-1200の実行例です。

バージョン番号、タイムスタンプ、プログラム容量、点データ数などを表示します。

#VER

MPC-1200H BL/I 1.14_29 2014/08/08

All Rights reserved. ACCEL Corp.

PRG_430K PNT_20K DIM_20K .T32

ERR\$

書式

ERR\$(n)

使い方

PR ERR\$(err_)

機能

エラーコードに対応するメッセージの出力

解説

ON_ERROR でエラー割り込みを設定すると、err_変数にエラーコードと対応する文番号が格納されます。上位1byteがエラー・コード、下位3byteが文番号です。

ERR\$()はこの上位1byteのコードにしたがってエラー文字列を返します。

したがって、手操作でエラー・メッセージを参照する場合は、

PRINT ERR\$(1>>24)

というように値を24bite上位にシフトさせます。

BATTERY

書式

BATTERY

使い方

```
IF BATTERY != 0 THEN  
  MBK(20)=BATTERY
```

機能

電池状態

解説

電池を搭載している機種で、電源オフ時に電池電圧が低下すると、パワーオン直後にBATTERY OUTもしくはBATTERY LOWという表示が出ます。BATTERY OUTは電池完全消耗か電池自体が抜き取られたことを意味します。BATTERY LOWは電池電圧低下です。BATTERY OUTの場合は、予約変数BATTERYが1、BATTERY LOWの場合は2となります。

IO

Category

CLR_OUTP

書式

CLR_OUT arg

使い方

CLR_OUTP 1|8
CLR_OUTP 15

機能

IOエリアの初期化

解説

CLR_OUTP [n]

n=1: 実出力ポート

2: CUNET

4: MBK

8: Memory IO

ビットパラメータのため必要な初期化エリアに対応するビットをONして実行します。

CLR_OUTP 15 は全エリアの初期化となります。

FLIP_FLOP

書式

FLIP_FLOP o_port IN(port) [pat]

使い方

FLIP_FLOP -1 IN(24)
FLIP_FLOP -1 IN(24) &H0F

機能

セトリセットブリップフロップ

解説

セトリセットタイプのフリップ・フロップです。8ビットのバンク単位のI/Oで設定できます。
実行内容としては、

$o_port \mid= IN(n) \text{ xor } pat$

となります。

patが省かれているとpatは0になります。

このため、入力ポートがアクティブになると、対応する出力ビットがセットされ保持されます。

クリアするには、OFF bit_portします。

セットに必要な時間は1msecです。

ネガティブ論理が必要な場合は、patの対応ビットを1にします。

```
10 CLR_OUTP 15
20 FLIP_FLOP -1 IN(24)
30 DO
40 PRX IN(24) IN(-1)
50 TIME 500
60 LOOP
#RUN
```

00000000 00000000 SW(194)=0,SW(193)=0,SW(192)=0

00000001 00000001	SW(194)=0,SW(193)=0,SW(192)=1
00000000 00000001	
00000002 00000003	SW(194)=0,SW(193)=1,SW(192)=0
00000000 00000003	
00000004 00000007	SW(194)=1,SW(193)=0,SW(192)=0
00000000 00000007	

LATCH

書式

```
LATCH i SW(n) IN(m) j  
LATCH i @SW(n) IN(m) j
```

使い方

```
LATCH 1 SW(208) IN(24) 2  
LATCH VOID
```

機能

15bit ラッチ

解説

SW(n)をトリガとして、IN(m),IN(m+1)の平行値をバンクj,j+1に複写します。
ただしj+1バンクのMSBは、ラッチマークに使用されるので、有効データは15bitです。
@SW(n)とすると論理反転入力になります。

トリガは1msec幅以上のON信号。
iは0～3まで指定可能なため、4種類のラッチを設けることができます。

LATCH

VOIDですべてのラッチ設定を解除できます。また、パワーオンリセットでもラッチ機能は無効になります。
LATCHコマンドは引数無で現在のラッチ機能設定状態を確認することができます。

サンプルプログラムではそれぞれのトリガ信号に対して同じ入力バンクから、それぞれ異なるメモリIOにデータを複写します。

ハンドシェークは、j+1バンクのメモリI/OのMSBで行っています。
なお、出力には使用していない出力ポートを指定して用いることもできます。

10 LATCH 0 SW(208) IN(24) -2

```
20 LATCH 1 SW(209) IN(24) -4
30 LATCH 2 SW(210) IN(24) -6
40 LATCH 3 SW(211) IN(24) -8
45 DO
50 IF SW(-8)==1 THEN : PRINT "latch 0" : WAIT SW(208)==0 : OFF -8 : END_IF
60 IF SW(-24)==1 THEN : PRINT "latch 1" : WAIT SW(209)==0 : OFF -24 : END_IF
70 IF SW(-40)==1 THEN : PRINT "latch 2" : WAIT SW(210)==0 : OFF -40 : END_IF
80 IF SW(-56)==1 THEN : PRINT "latch 3" : WAIT SW(211)==0 : OFF -56 : END_IF
90 LOOP
```

OFF

書式

OFF arg1 [arg2 arg3 arg4 ...]

使い方

OFF 1 2 3 /* MIO-1616etc

OFF A A+1

OFF -1 /* Memory I/O エリア

機能

出力ポートのOFF

解説

出力ポートをOFFします。オープンコレクタ出力がフLOAT状態になります。

負の値の場合(-1 ~)は、メモリI/Oエリアのビットオフです。

2000以上の場合(2000 ~)は、CUnetエリアのビットオフです。

70000以上の場合(7aabb)は、MBK I/Oエリア(タッチパネルRエリア)のビットオフです。

aaがバンク番号(0 ~ 99)でbbはビット番号で0 ~ 15の値になります。

ON

書式

ON arg1 [arg2 arg3 arg4 ...]

使い方

ON 1 2 3 /* MIO-1616 etc
ON A A+1
ON -1 /* Memory I/O
ON 2000 /* CUnet Area
ON 70000 /* MBK I/O area

機能

出力ポートのON

解説

出力ポートをONします。オープンコレクタ出力がシンク状態になります。

負の値の場合(-1 ~)は、メモリI/Oエリアのビットオンです。

2000以上の場合(2000 ~)は、CUnetエリアのビットオンです。

70000以上の場合(7aabb)は、MBK I/Oエリア(タッチパネルRエリア)のビットオンです。

aaがバンク番号(0 ~ 99)でbbはビット番号で0 ~ 15の値になります。

OUT

書式

OUT val port
OUT val port1,port2..
OUT val port1 TO port2

使い方

OUT &H55 2
OUT &HAA -1
OUT 0 1 2 5
OUT 0 -1 TO -10

機能

出力ポート、メモリI/Oを8bitパラレル設定します。

解説

出力ポートを1byte設定するコマンドで、バンク指定となります。

MPC-2000のI/Oポート0～7はバンク 0、8～15は、バンク1となります。

1枚目のMIO-1616は、同様にバンク2、3が割り当てられます。バンク設定を負の値とするとメモリI/Oとなります。

アドレス値に-Lng、-Wrd、-Intを与えると、それぞれロング書き込み、整数2byte書き込みとなります。

書き込みにはWrd,Intの区別はありません。タッチパネルのI/Oエリア(Rエリア)は70000以上を指定します。
abは00～99となります。

OUT data 7ab00 /* byte書き込み

OUT data 7ab00-Ub /* Hi-byte書き込み

OUT data 7ab00-Wrd /* ワード書き込み

OUT data 7ab00-Lng /* ロング書き込み

また、出力ポートを複数同じ値に設定する場合は、出力ポート番号を続けて記述します。
連続して同じ値とするときは、port1 TO port2 という記述をします。

PULSE_OUT

書式

PULSE_OUT port# interval [count]

使い方

```
PULSE_OUT 0 10 10
PULSE_OUT 0 10
PULSE_OUT VOID
PULSE_OUT 32767
```

機能

出力ポートの自動オンオフ

解説

出力ポートを自動でオンオフします。

countを指定すると指定回数ON/OFFの後OFFとなります。

intervalは0.1秒単位設定。

コマンドが実行されると内部カウンタが初期化され、指定時間後(Interval/2)、最初にONコマンドが実行されます。

このため予め指定ポートをONにしておくと、interval後にOFFとなります。

ON/OFF中のポートを停止するには、intervalを0にします。この時指定ポートはOFFにします。

PULSE_OUT 0 10 ではポート0を1秒間隔でON/OFF。

PULSE_OUT 0 0 でON/OFFを停止します。

PULSE_OUT VOID は、すべての設定されたPULSE_OUTをキャンセルします。

PULSE_OUT 32767 は すべての設定されたPULSE_OUTの動作を同期させます。

PWM

書式

PWM portn k

使い方

PWM 15 A

機能

PWMパルス発生

解説

指定された出力ポートをPWMオンオフします。
PWM周期時間は50msecです。与えられたk msecだけポートをオンします。
発熱体や、ペルチェ素子の電力制御に用います。

*PWMはPulse Width Modulationの略式表現です。

SENSE_ON,SENSE_OFF

書式

SENSE_ON port sw

使い方

SENSE_ON 16 -1

機能

リアルタイムON/OFF

解説

指定した入力が入力になると、指定ポートをリアルタイム(1msec以内)でONします。(OFFの場合はSENSE_OFF)

SENSE_ON 16 192

SW(192)がオンすると、16をオンします。

SENSE_OFF 16 192

SW(192)がオンすると、16をオフします。一度反応すると、設定は解除されます。

また、強制解除は SENSE_ON VOID です。

@SW

書式

@SW(arg)

使い方

IF (@SW(0)&SW(1))==1 THEN

機能

入力ポートの反転読み取り

解説

SWの値を論理反転して返します。

```
LIST
10 ON -1 -3 -5
20 PRINT @(SW(-1))|@(SW(-3))|@(SW(-5))
30 ON -1 -3 -5
40 PRINT @SW(-1)|@SW(-3)|@SW(-5)
50 PRINT SW(-1)|@SW(-3)|@SW(-5)
#run

*
Compiling
-----
0
0
1
#
```

CSW

書式

CSW(arg)

使い方

```
A=CSW(0)
IF A==1 THEN : GOSUB *A :ELSE : GOSUB *B
```

機能

指定入力ポートが変化するまで待ち、変化後の値を返す。

解説

CSW(n)は関数自身が待ち(ポーリング)を含んでいます。
下のサンプルでは入力の状態が変化するまで50で待ち続けます。

```
10 FORK 1 *task1
20 END
30 *task1
40 DO
50 A=CSW(-1)
60 PRINT A
70 LOOP
#run

#on -1
#off -1
#0
on -1
#1
off -1
#0
```

HIN

書式

HIN(arg)

使い方

A=HIN(24)

A=HIN(24~Wrd)

機能

8ビットパラレル入力 ~Lng,~Wrd等の型指定をいれるとロング、ワードパラレルとして読み取れる

解説

HIN()は、IN()と同じ機能関数ですが以下の相違があります。

IN()はIOエリアの入力ポートを2度読みしてチャタリングノイズを防いでいます。

これに対してHIN()は、1回読みです。

メモリI/Oなどの外部入力でない領域の読み取りは、HIN、INとも1度読みです。

HSW

書式

HSW(arg)

使い方

```
A=HSW(192)
IF HSW(192)&HSW(200)&HSW(208) THEN
```

機能

入力ポートのビット読み取り

解説

SW()は入力ポートに実入力ポートを指定すると2度読みをします。これに対してHSW()は1度読み取りのみです。

```
IF HSW(192)&HSW(200)&HSW(208) THEN
```

このように複数のSWの論理をとる場合、SW()では2度読みが発生し実行速度が遅くなってしまいます。上の例のようにHSW()を使用すれば、高速で論理演算を行います。

出力ポート、メモリI/O、MBKのI/Oエリアのモニタでは、この区別はありません。

IN

書式

IN(arg1)

使い方

```
IF IN(0)==&HAA THEN  
WAIT IN(1)==&H05  
A=IN(0-Lng)
```

機能

入力ポートの平行取り込み(8bits)

解説

MPC-2000 入力ポートは24,25

最初のMIO-1616の入力ポートは26,27となります。負の数を指定すると、メモリI/Oとなります。

アドレス値に~Lng,~Wrd,~Intを与えると、それぞれロング読み取り、整数2byte読み取り、符号付2byte読み取りとなります。

タッチパネルのmbkエリアは、70000以上を指定します。

abは00 ~ 99となります。

IN(7ab00) byte読み取り

IN(7ab00-Ub) hi-byte読み取り

IN(7ab00-Wrd) ワード読み取り

IN(7ab00-Lng) ロング読み取り

DSW=IN(24)/16 /* GET DSW value and Shift down 4bits

M_SW

書式

M_SW([n],[n])

使い方

M_SW(192)
M_SW(10,193)

機能

フィルタ付きSW関数

解説

メカスイッチや反射センサなどチャタリング信号を発生しやすい入力に対して使用するSW()関数です。
M_SW(n)の場合は、1msecごとに三回、nポートを読み取り三回とも同じ値の場合の時のみポート値を返します。
M_SW(t,n)の場合は、tが読み取り回数指定となり、t回(tmsec)同じ値の場合に、ポート値を返します。
従って、入力が1msec周期でパルス状に変化している場合には、M_SW()関数はサスペンドとなります。
指定できるポート番号nはボード上の入出力I/Oのみで、メモリI/Oなどには使用できません。

SW

書式

SW(arg)

使い方

```
A=SW(192) /*入力ポートの読み出し  
IF SW(A)==0 THEN : ON 5 : END_IF /*入力による条件分岐  
WAIT SW(192)==1 /*条件待ち
```

機能

入力ポートの読み出し

解説

入力ポートがGNDにショートされると1を返します。
フロート状態では0となります。

WS0,WS1

書式

WS0(arg1)

使い方

```
IF WS0(0)==1 THEN : GOTO *TMOUT : END_IF
```

機能

タイムアウト付I/O待ち関数

解説

WS0(n) はSW(n)が0になるのを待ちますが、TMOUTで設定した待ち時間を越えると、値1を返します。時間内に0となったら、0を返します。

WS1(n) はSW(n)が1になるのを待ちますが、TMOUTで設定した待ち時間を越えると、値1を返します。時間内に1となったら、0を返します。

注)WS0,WS1はtimer_を使用をします。このため、WS0,WS1を実行する上位処理でtimer_を使用したタイムアップ監視を行っている場合は、WS0,WS1内でtimer_の控えをとり、WS0,WS1から抜け出るときにtimer_を戻します。このため、おおむね、矛盾なく動作しますが、WS0から抜けるごとに1デジット(0.1秒)程度の誤差が生じます。

AUTO_RESET_1

書式

AUTO_RESET_1

使い方

AUTO_RESET_1=1000

機能

オートリセット

解説

参照 COUNTER_1

AUTO_RESET_2

書式

AUTO_RESET_2

使い方

AUTO_RESET_2=500

機能

オートリセット

解説

参照 COUNTER_1

COUNTER_1

書式

COUNTER_1

使い方

COUNTER_1=0

機能

カウントアップ

解説

カウント機能

それぞれの入力ポートへの1msecパルス幅以上に対して、カウントします。

*MPC-1200/MPC-1000/MPC-N816

SW(198) --> COUNTER1

SW(199) --> COUNTER2

*MPC-2000では、

SW(206) --> COUNTER_1

SW(207) --> COUNTER_2

カウント開始は初期値を設定した時点となります。それまでの初期値はVOIDでカウントは無効になっています。

オートリセット機能

予約変数AUTO_RESET_1に値を設定すると、その値になるとCOUNTER_1は0クリアします。

予約変数AUTO_RESET_2に値を設定すると、その値になるとCOUNTER_2は0クリアします。

アップダウンカウント

AUTO_RESET_1を0にすると、COUNTER_1 がアップダウンになり、COUNTER_2は無効となります。

COUNTER_2

書式

COUNTER_2

使い方

COUNTER_2=0

機能

カウントアップ

解説

参照 COUNTER_1

AVOID

書式

AVOID

使い方

CONST sol1 AVOID

機能

I/Oコマンド無効化

解説

I/Oコマンドの無効化。サンプルプログラムでは、sol1をAVOIDとして、sol1に対するONコマンドを無効としています。

```
10 CONST sol1 AVOID /* not use
20 CONST sol2 1
30 ON sol1 sol2 /* sol1 disable, sol2 enable
```

通信

Category

CNFG#

書式

CNFG# COMn [RS485] "setting"

使い方

CNFG# 1 "38400b8pns1NONE"
CNFG# 5 RS485 "38400b8pns1NONE"

機能

RS-232Cポートの初期化

解説

COMn は初期化する RS-CH 番号です。文字列はボーレート、キャラクタフォーマット指定です。

COMn = 1 MPC-USER ch1

COMn = 2 MPC-USER ch2

COMn = 3 MRS CH3

COMn = 4 MRS CH4

COMn = 5 MRS CH5

COMn = 6 MRS CH6

COMn = 7 MRS CH7

COMn = 8 MRS CH8

ボーレート 4800,9600,19200,38400 のいずれかを選択します。

b8 8 ビットキャラ

b7 7 ビットキャラ

pn パリティなし

pe 偶数パリティ

po 奇数パリティ

s1 1stop bit

s2 2 stop bit

NONE XON/XOFF 制御なし (XON/XOFF 制御未対応)

引数にRS485を追加すると、RS422/485兼用ポートでRS485通信が可能になります。

プログラム実行中にバッファクリア等の目的でCNFG#を実行すると、字落ちの原因となります。
CNFG#は初期設定で一回のみ使用として、バッファクリアにはINPUT#とCLR_BUF定数を併用してください。

参照 INPUT#、PRINT#

INPUT

書式

INPUT [CH] [EOL|x] [CHR_C|x] [TMOUT|x] a\$

使い方

INPUT a\$

機能

文字列入力

解説

INPUTは、INPUT#でCH0(プログラムポート)に固定されたシリアル入力コマンドです。
使い方は、INPUT#と同様ですので、INPUT#を参照してください。

INPUT#

書式

```
INPUT# [CH] [EOL|x] [CHR_C|x] [TMOUT|x] a$  
INPUT# [CH] CLR_BUF
```

使い方

```
INPUT# a$  
INPUT# CH a$  
INPUT# 5 EOL|10 c$  
INPUT# 3 CHR_C|54 a$  
INPUT# 3 TMOUT|10 a$  
INPUT# 20 a$  
INPUT# 2 CLR_BUF  
INPUT# 5 COMPOWAY rcv$
```

機能

RS-232Cポートより文字列を取り込む。

解説

INPUT#はシリアルポートより文字列を取り込みます。CH番号を省略すると、CH1になります。
デフォルトでターミネータはCRとなっていますが、EOL|xx オプションで変更できます。xxにはアスキーコード。
文字カウント取り込みの場合は、CHR_C|xxオプションを用います。xxは255以内の数値。
カウントを指定するとターミネータは無視されます。
タイムアウトを必要とする場合は、TMOUT|xx オプションを用います。xxには秒単位で制限時間をいれます。
TMOUT|10の場合10秒以内でよみとれない場合は処理を打ち切ります。
タイムアウトしたかどうかは、rse_変数を参照します。rse_が1の場合タイムアウトとなったことを意味します。
CLR_BUFを引数に与えた場合は、バッファの文字列をすべて読み捨てます。
なお、オプションパラメータとしてCOMPOWAYを与えると、OMRON
COMPOWAYフォーマットでの受信になります。
TMOUTオプションも併用できます。チェックサムエラー発生の場合は、rse_が4となります。
COMPOWAYフォーマットで受け取った文字列は、COMPOWAYコマンドで基本分解することができます。

数値変換は、VAL関数,GET_VALなどの文字列処理コマンドで行います。

INPUT# a\$

a=VAL(a\$) : b=VAL(0) 関数VALを参照してください。

--Serial Communication--

CNFG# 3 "38400b8pns1NONE"

CNFG# 4 "38400b8pns1NONE"

CNFG# 5 "38400b8pns1NONE"

a\$="123456789012345678abcdefghijklmnopqrstuvwxy\$%&()01234"

' GOTO *RS422

DO

PRINT# 3 a\$ " /r"

INPUT# 4 EOL|13 b\$

PRINT# 4 b\$ " /n"

INPUT# 5 EOL|10 c\$

PRINT# 5 c\$

INPUT# 3 CHR_C|54 a\$

PRINT a\$

LOOP

*RS422

DO

PRINT# 4 a\$ " /r"

INPUT# 5 b\$

PRINT b\$

PRINT# 5 b\$ " /r"

INPUT# 4 a\$

LOOP

PRINT#

書式

PRINT# [COM#] [Options] arg1 arg2 ...

使い方

```
PRINT# 1 a$ "123 /n"  
PRINT# 5 COMPOWAY snd$  
PRINT# 3 STR_LEN|32 a$
```

機能

通信ポートへ出力

解説

PRINT# はシリアルポートへの出力です。

最初に引数が数値である場合、その数値はRS-CH番号指定となります。

出力引数として文字列、文字列変数、変数などが使えます。

```
PRINT# "count=" i_ " " i_ *i_
```

なお、PRINT#では引数の出力間にスペース挿入はありません。

また、引数の上で、+による文字列結合はできませんが、

以下のように、引数を羅列するだけで、文字列結合して出力するのと同じことができます。

```
PRINT# CHR$(1) "DATA" CHR$(3)
```

よって以下と同じ結果になります。

```
b$=CHR$(1)+"DATA"+CHR$(3)
```

```
PRINT# b$
```

固定長出力オプション STR_LEN

文字列出力は、通常NULLターミネーションされます。しかし、バイナリコードを含む

固定長文字列出力が必要となる場合があり、STR_LENオプションはこうした場合に用います。

```
a$="1234567" : b$="abcdfge"
```

```
print# STR_LEN|4 a$ b$
```

この場合、出力されるのは、1234abcd となります。

NULLコードを含む文字列の出力

NULLコード、つまりアスキーコード0は、通常文字列のターミネータと扱われており、通常の方法では出力されません。

1) 簡単に0～4のコードを出力するには、文字列定数中に /0～/4を記述します。

PRINT# "ABC /0DEF" --> ABC-00DEF ABCとDEFの間に00コードが出力されます。

2) チェックサムを出力する方法1

例えば16bitのチェック・サムを出力する場合は以下のようにします。

```
HI=CHK_SUM>>8
```

```
LO=CHK_SUM&255
```

```
PRINT# STR_LEN|2 CHR$(HI) CHR$(LO)
```

3) チェックサムを出力する方法2

文字列の固定パケットに直接バイナリコードを埋め込む方法です。

```
CMND$="CMNDEXE"
```

```
SUM=0
```

```
FOR i=0 TO LEN(CMND$)-1 : SUM=SUM+PEEK(CMND$+i) : NEXT
```

```
HI=SUM>>8
```

```
LO=SUM&255
```

```
POKE 0 HI LO (CMND$+7)
```

```
PRINT# STR_LEN|10 CMND$
```

*POKEコマンドは、直接メモリにデータを書き込みます。記述に誤りがあると、誤作動やプログラム破壊を引き起こします。

注意して使用してください。

【Optionsについて】

COMPOWAY:

定数COMPOWAYを与えると、OMRON COMPOWAYフォーマットで文字列を出力します。

転送する文字列はコマンドCOMPOWAYであらかじめパケット化しておきます。

STR_LEN:

定数STR_LENに転送文字数をORすると(例:STR_LEN|32)文字列出力は、

ヌルターミネータは無視され指定された転送文字数 出力します。

ヌルコードを含む転送に使用します。

ヌルを含む文字列の作成には、コマンド、ADD_STRを使用します。

--- Examples---

```
PRINT# 1 "ABC /r" /* Xmit "ABC[CR]" through CH1
PRINT# 1 "ABC /n" /* Xmit "ABC[LF]" through CH1
PRINT# 1 "ABC /r /n" /* Xmit "ABC[CR][LF]" through CH1
PRINT# 1 "ABC /tDEF" /* Xmit "ABC[TAB]DEF" through CH1
```

```
/r=[CR]=&H0D
/n=[LF]=&H0A
/t=[TAB]=&H09
```

--- An example of COMPOWAY---

```
COMPOWAY node_no sub_adr sid cmnd_txt$ snd$
PRINT# 5 COMPOWAY snd$
```

LOF

書式

LOF(ch)

使い方

```
IF LOF(1)>10 THEN :input# 1 a$: END_IF
```

機能

バッファの文字列数を返す。

解説

各RS-232Cポートのバッファにたまった文字数を返します。引数のCHは0～11でに対応。
また、LOF(20)の場合はUSBメモリの残文字の有無で、1で有、0でEOFまで達したことを意味します。

PRX LOF(-1)とするとMRS-MCOMのバージョンを表示します。
-1はCH3～CH5、-2はCH6～CH8、-3はCH9～CH11を指定します。

```
#PRX LOF(-1)
```

```
20090514
```

MRS-MCOM6では、電源が投入されていないとこの値が0となります。

rse_

書式

rse_

使い方

pr rse_

機能

通信エラーステータス

解説

タスク変数

通信時のエラー内容を表示します。

```
10 CNFG# 1 "9600b8pns1NONE"
20 INPUT# 1 TMOUT|5 a$ /* timeout 5 sec
35 IF rse_==1 THEN /* check timeout
36 PRINT "timeout"
37 ELSE
38 PRINT a$
40 END_IF
#RUN

timeout /* fail
#RUN

asdfg /* success
#
```

CHR_C

書式

CHR_C

使い方

INPUT# 1 CHR_C|1 a\$

機能

受信文字数設定

解説

受信文字数を設定します。

```
10 CNFG# 1 "9600b8pns1NONE"  
20 INPUT# 1 CHR_C|1 a$ /* receive 1 character  
30 PRINT a$  
#RUN
```

a /* send 'a' from the terminal soft

CLR_BUF

書式

CLR_BUF

使い方

INPUT# 1 CLR_BUF

機能

RS-232入力バッファクリア

解説

INPUT#のオプションです。入力バッファをクリアします。
バッファのクリアにはCNFG#を用いないでください。

INPUT#参照

EOL

書式

EOL

使い方

INPUT# 1 EOL|10 a\$

機能

受信ターミネータ設定

解説

受信ターミネータを設定します。

```
10 CNFG# 1 "9600b8pns1NONE"  
20 INPUT# 1 EOL|10 a$ /* until receive LF(&HA=10)  
30 PRINT a$  
RUN
```

```
hello /* send 'hello' from the terminal soft
```


STR_LEN

書式

STR_LEN

使い方

PRINT# 3 STR_LEN|32 a\$

機能

出力文字数指定

解説

PRINT#参照

TMOUT

書式

TMOUT

使い方

INPUT# 1 TMOUT|5 a\$

機能

受信待ちタイムアウト設定

解説

INPUT#コマンドの受信待ちには時間制限がありませんが、
TMOUTオプションにより、制限時間を設けることができます。
時間制限を越えた場合、rse_変数にTMOUTの発生の有無が反映されます。

```
10 CNFG# 1 "9600b8pns1NONE"
20 INPUT# 1 TMOUT|5 a$ /* timeout 5 sec
35 IF rse_==1 THEN /* check timeout
36 PRINT "timeout"
37 ELSE
38 PRINT a$
40 END_IF
#RUN

timeout /* fail
#RUN

asdfg /* success
#
```

パルス発生

Category

ACCEL

書式

ACCEL [axis] PPS [leng,lo_pps]

使い方

```
ACCEL 4000
ACCEL 4000 1000 100
ACCEL Z_A 8000
ACCEL SACL 4000
ACCEL X_A|SACL 2000
ACCEL X_A|OUTSL 30000
```

機能

加速度設定

解説

PGの加速度を設定します。軸指定を省略すると、全軸に適用されます。

指定パラメータは、最大速度(pps)、加速距離(pulse),自起動(pps)です。

加速距離以下を省略するとデフォルト値を設定します。

このコマンドはパルス発生中には使用できません。パルス発生中はSPEEDコマンドを用いてください。

また、自起動速度を低く設定すると、全体の動作が遅くなります。

ステップモータでも100pps程度、サーボモータでは1kpss程度を最小速度の目安としてください。

(1ppsを最小速度とすると、1パルスの出力に1秒必要になります。もし、動作の最初と最後に1ppsが発生すると、それだけで二秒ずつ必要になります。)

軸指定パラメータに定数SACLをORすると、S字加減速となります。

例: ACCEL X_A|SACL 80000

軸指定パラメータに定数OUTSL

をORすると、RANGE設定値と現在位置が比較され、出力ポートに反映されます。(MPG-2314 CEP128D以降)

RANGE X_A 10000 XXXの場合 X(0)が9999まではO0がOFF,10000以上になるとON

RANGE X_A -10000 XXXの場合 X(0)が-10001まではO0がOFF,-10000以上になるとON

*O0は、MPG-2314 J4-19です。

なお、引数無しで実行すると、設定パラメータと、設定したACCELの文番号を表示します。文番号が0の場合はプログラムによっては設定されていないことを意味します。

#accel

X=> Max=3000 Length=150 Min=300 Feed=100 Set@20

Y=> Max=3000 Length=150 Min=300 Feed=100 Set@30

U=> Max=8000 Length=400 Min=800 Feed=100 Set@0

Z=> Max=3000 Length=150 Min=300 Feed=100 Set@40

#

BACKLASH

書式

BACKLASH Xb Yb Ub Zb

使い方

BACKLASH 111 121 0 0

機能

バックラッシュ補正設定

解説

MPG-2314のパルス出力に、バックラッシュ補正を与えるものです。

バックラッシュ補正は、単軸、直線補間のみ有効です。円弧補間には適用されません。

パワーオンリセット後は、0となっており、電源切断後、都度設定が必要となります。

バックラッシュ補正は、パルス発生方向の変わり目でバックラッシュ設定されたパルスが加算されるものです。

使い方には注意が必要で、機械系のバックラッシュ状態を初期化しておく必要があります。

たとえば、原点復帰後、CW方向にバックラッシュ量以上にダミー移動させてバックラッシュ値を正の値に設定します。

パルス発生方向がバックラッシュ値と同じである限り、バックラッシュパルス加算は行われませんが、

パルス発生が負方向になったときに、バックラッシュ値を負の値に変換し、パルス加算を行います。

したがってバックラッシュ値は内部で、動作によって正負に反転させられており、方向監視もかねています。

なお、バックラッシュ補正は、万能ではありません。

機械系のバックラッシュ量は、移動速度、負荷、振動などの条件によって変動します。

機械系の特性をよく把握した上で使用してください。

CLRPOS

書式

CLRPOS [AXIS],[**-1**]

使い方

CLRPOS
CLRPOS X_A
CLRPOS -1
CLRPOS X_A -1

機能

位置カウンタ、エンコーダカウンタのクリア

解説

引数がない場合は現在位置をすべて0にします。

軸指定定数があれば、対象軸を0にします。**-1**が与えられると、エンコーダ・カウンタを0にします。

CLRPOS X_A -1の場合は、X軸エンコーダ・カウンタをクリアします。

CP

書式

CP

使い方

機能

現在位置表示

解説

プログラムのデバッグ用コマンドです。
MPGボードの座標管理での現在位置を表示します。

DS_DACL

書式

DS_DACL [axs]

使い方

DS_DACL
DS_DACL X_A

機能

減速無効設定

解説

自動減速無効とする。連続補間で使用。

EN_DACL

書式

EN_DACL [axs]

使い方

EN_DACL
EN_DACL X_A

機能

減速有効設定

解説

自動減速有効とする。

FEED

書式

FEED [axis] n
FEED fx fy fu fz

使い方

FEED 10
FEED X_A 100

機能

速度設定

解説

ACCELで設定された最高速、最低速をもとに100段階に速度を設定します。

数値は最高速の%を整数値で設定します。

FEED X_A 100で、X軸最高速

FEED X_A 0でX軸最低速度。

途中の数値は、 $F_n = \text{MIN} + n * ((\text{MAX} - \text{MIN}) / 100)$

軸パラメータがない場合は、X,Y,U,Zの順序で指定パラメータとします。

したがってFEED 100はX軸に対してのみ速度を指定したことになります。

FLP

書式

FLP

使い方

機能

フラッシュROM読込

解説

MPC-1000/MPC-N816専用コマンド

フラッシュROMから点データP(100) ~ P(299)を読み込みます。

このエリアはパワーオンリセット時に自動的に読み込まれますが、FLPコマンドでも読み込みができます。

フラッシュROMへ書き込むのはFSPコマンドです。

```
10 FOR I=100 TO 299
20 SETP I I I+1 I+2 I+3
30 NEXT I
40 FSP
50 NEWP
60 PRINT "P(100)=" P(100) "P(299)=" P(299)
70 FLP
80 PRINT "P(100)=" P(100) "P(299)=" P(299)
#RUN

P(100)= 0 0 0 0 P(299)= 0 0 0 0
P(100)= 100 101 102 103 P(299)= 299 300 301 302
```

FSP

書式

FSP

使い方

機能

フラッシュROM書き込み

解説

MPC-1000/MPC-N816専用コマンド

点データP(100)～P(299)をフラッシュROMに書き込みます。

このエリアはCompiling後のプログラムとともにフラッシュROMに書き込まれますが、プログラム中で強制的に書き込む場合はFSPコマンドを用います。

MPC-1000はバッテリーバックアップがありません。

P(100)～P(299)は電源を切っても保持したいデータのエリア(ティーチングポイント、バックアップ変数等)として使用します。

フラッシュROMから読み込むのはFLPコマンドです。

```
10 FOR I=100 TO 299
20 SETP I I+1 I+2 I+3
30 NEXT I
40 FSP
50 NEWP
60 PRINT "P(100)=" P(100) "P(299)=" P(299)
70 FLP
80 PRINT "P(100)=" P(100) "P(299)=" P(299)
#RUN
```

P(100)= 0 0 0 0 P(299)= 0 0 0 0
P(100)= 100 101 102 103 P(299)= 299 300 301 302

HOME[MPC-1200]

書式

HOME X Y U Z
HOME axs V

使い方

HOME X_A NEG_L

機能

原点復帰コマンド

解説

MPC-1200の入力ポート(SD,ORG)を使用した原点復帰コマンドです。SD入力はスローダウン、ORG入力は即停止です。

パワオン後はSD無効です。SHOMでSD有効化の場合、SDオン後ORGが有効になります。

```
PG 17
ACCEL X_A|Y_A 20000 2000 500
FEED X_A|Y_A 100
IF SW(200)==1 THEN ' check ORGX
RMVS X_A 5000
END_IF
IF SW(201)==1 THEN ' check ORGY
RMVS Y_A 5000
END_IF
WAIT RR(X_A|Y_A)==0
FEED X_A|Y_A 25
SHOM &H50
HOME X_A|Y_A NEG_L
```

HOME[MPG-2314]

書式

HOME X Y U Z
HOME axs V

使い方

HOME 10000 10000 1000 1000
HOME NEG_L NEG_L NEG_L NEG_L
HOME X_A -1000
HOME X_A|Y_A -1000

機能

原点復帰コマンド

解説

SHOMで設定された停止条件を与えながらHOMEシーケンスを実行します。
HOMEコマンドにはタイムアウトが有効になっていますので、適切に設定します。
タイムアウトで停止した場合は、現在値をクリアしません。X(0)等が0でなければ、エラー停止です。

HOMEコマンドの引数は、ニアオリジンサーチのための移動量です。移動中にニアオリジンを検出すると減速停止します。

SHOMでIN1_ON/IN1_OFFが設定されていると、ニアオリジン検出後、ACCELで設定した最低速度でZ相サーチとなります。

このときの移動方向は、SHOMで与えたCWもしくはCCWで決定されます。デフォルトは、CCWとなっています。
プログラム1はタイムアウト10秒で原点復帰の例です。HPT(XIN0),HPT(YIN0),HPT(ZIN0)は、各軸のIN0のモニターです。

ニアオリジンの内側にいると、退避移動をしてからの原点復帰となります。

なお、原点復帰ニアオリジンの移動量を大きくとりたい場合は、POS_L,NEG_Lを用いてください。
これらは、正・負の3byte長の最大数です。
ニアオリジン検出移動量は、HOME X_A -1000 のように軸指定定数を使用することもできます。

----program1---

10 PG 1

20 ACCEL 40000

30 ACCEL Z_A 20000

40 SHOM X_A|Z_A|Y_A IN0_ON|IN1_ON|CW

50 IF HPT(XIN0)==0 : RMVS X_A 20000 : END_IF

60 IF HPT(YIN0)==0 : RMVS Y_A 20000 : END_IF

70 IF HPT(ZIN0)==0 : RMVS Z_A -20000 : END_IF

80 WAIT RR(ALL_A)==0

85 TMOUT 10000

90 HOME -100000 -100000 0 100000

---XY Robot example---

*HOME1

ACCEL X_A|Y_A 10000 100 100

IF HPT(XIN0)!=0 THEN

RMVS X_A 10000

END_IF

IF HPT(YIN0)!=0 THEN

RMVS Y_A 10000

END_IF

WAIT RR(ALL_A)==0

TIME 100

SHOM X_A|Y_A IN0_ON

HOME -100000 -100000 0 0

WAIT RR(ALL_A)==0

TIME 100

RMVL 2000 2000 0 0

WAIT RR(ALL_A)==0

STPS 0 0 VOID VOID

PRINT "XY HOME"

TIME 100

RETURN

HOUT

書式

HOUT arg

使い方

HOUT 1

機能

MPGボードの出力ポートの制御

解説

MPG-2314出力ポートの4ビットの一括設定です。

H_OFF

書式

H_OFF arg

使い方

H_OFF 2

機能

MPG-2314ボードの出力ポートオフ

解説

通常の出力ポート同様ビットオフします。

H_ON

書式

H_ON n

使い方

H_ON 1

機能

MPG-2314の出力ポートのオン

解説

通常のポート同様ビットオンします。

INCHK

書式

INCHK

使い方

機能

MPG-2314の入力状態のモニタ

解説

INCHKと入力すると入力ポートの状態を表示します。'q'をタイプすると停止します。

```
inchk
MPG-2314
X=+LMT:off-LMT:off ALM:off INP:off IN0:off IN1:off
Y=+LMT:off-LMT:off ALM:off INP:off IN0:on IN1:off
U=+LMT:off-LMT:off ALM:off INP:off IN0:off IN1:off
Z=+LMT:off-LMT:off ALM:off INP:off IN0:off IN1:off
#
```

INSET

書式

INSET [axs] Settings

使い方

INSET PHASE4
INSET ALL_A ALM_ON|INP_OFF

機能

MPG-2314用入力設定コマンド

解説

入力ポートの機能を設定します。機能と予約定数との関係は以下のとおりです。

INPOS => INP_ON,INP_OFF,INP_NO

ALARM => ALM_ON,ALM_OFF,ALM_NO

LMT => LMT_ON,LMT_OFF

ソフトリミット=> SLMT_ON,SLMT_OFF

エンコーダ => UP_DWN,PAHSE1,PAHSE2,PAHSE4

PLS => MD_2PLS,MD_DPLS(パルス発生モードの設定)

最後に実行したINSETが有効になります。

パラメータで与えられたもの以外の設定はリセットされます。

例)

INSET X_A ALM_ON|INP_ON

X軸に対して入力設定。アラームを有効にしてON状態をアラームとする。また、INPOSをONで有効とする。

INSET ALL_A ALM_ON|INP_OFF

すべての軸に対して入力設定。アラームはONで有効、INPOSはOFFで有効とする。

INSET PHASE4

エンコーダ入力を四通倍とする。

INSET ALL_A VOID

全設定クリア。RANGE設定(ソフトリミット)もクリア

INSET 【MPC-1200】

書式

INSET [axs] MD_DPLS

使い方

INSET MD_DPLS
INSET Z_A MD_DPLS

機能

MPC-1200用 パルス出力設定コマンド

解説

MPC-1200のパルス発生は、CW,CCWパルス方式がデフォルト設定となっていますが、INSETコマンドで方向指示型に変更することができます。方向指示に変更したい軸を指定して、MD_DPLSを引数を与えます。パワオンリセットでCW,CCW方式に戻ります。軸指定を省力すると全軸対象となります。

INTA_ON,INTB_ON

書式

INTA_ON portn (PG,axis)
INTB_ON portn (PG,axis)

使い方

INTA_ON 16 (0,X_A)
INTB_ON 17 (0,U_A)

機能

MPG-2314の割り込によるポートONもしくはOFF

解説

INTA_ONは、MPG-2314のカウンタ比較検出 割り込みによりポートをオンします。
割り込みを作動させるためには、以下のコマンド設定が必要です。

比較カウンタの設定 INSET axis CMP_PLS (もしくはCMP_CNT)
CMP_PLS = パルス位置 , CMP_CNT = エンコーダ・カウンタ位置
割り込みの有効化 STOP axis C_MORE (もしくはC_LESS)
C_MORE PIs >= COMP+ , C_LESS PIs 比較値COMP+の設定
RANGE axis VAL1 dummy

以上の設定があり、カウンタ値がVAL1を超えると(C_MOREの場合)割り込みが発生し、INTA_ONで指定されたポートをONします。(OFFの場合はINTA_OFFを用いる)

割り込みが発生したあとは、関数RR3(axis)を読み出すことによって割り込みは解除されます。

ただし、解除する前に、RANGE設定によって比較値COMP+を条件の外に変更しておく必要があります。

初期化

INSET axis CMP_CNT /*比較カウンタの選択
STOP axis C_MORE /*割り込み比較条件設定
RANGE axis VAL1 dummy
a=RR3(axis) /* 割り込みをクリアしておく

実行順序としては、

RANGE axis VAL1 summy /* 条件を変えておく

a=RR3(axis) /*割り込みクリア

SWAP

INTA_ON port /*割り込みポート設定

WAIT SW(port) /*割り込み発生を検出

となります。

*RR3のaxisは、X_A,Y_A等の単一軸指定のみ意味を持ちます。

割り込みの解除は、INTA_ON VOID あるいは、INTA_OFF VOIDを実行します。

なお、割り込みには、INTB_ON,INTB_OFFもあり、INTA_と合わせて2つのPGの割り込みまで対応することができます。

サンプルプログラムは、移動500パルスごとに割り込みを発生させ、外部にタイミングトリガを出力します。

INTA_ON,_OFF,INTB_ON,_OFFは、ソフトウェアのタイミング待ちと異なり、正確に位置タイミングを出力することができます。

```
INTA_ON VOID
PG 0
PG 0 1
PG 0 3
ACCEL 5000
CLRPOS
CLRPOS -1
INSET X_A CMP_PLS
DET_P=500
STOP X_A C_MORE
RANGE X_A DET_P 0
PG 0
FORK 1 *MPG
FORK 3 *MPG2
END
*MPG2
DO
INTA_ON 0 (0,X_A)
WAIT SW(0)
TIME 1
OFF 0
INC DET_P 500
```

```
RANGE X_A DET_P DET_P
A_=RR3(X_A)
TIME 5
LOOP
*MPG
RMVC X_A 1
END
```

JMPZ

書式

JMPZ Pnt

使い方

JMPZ P(n)

機能

Z下降なしJUMP

解説

ゲートモーション・コマンドJUMPの部分実行で、Z軸の下降を行いません。

JMPZは複数の動作が組み合わされた、複合コマンドです。このため、PAUSE、STOP、CONTを実行すると、思わぬところで横移動してしまうことがあります。

これを防ぐためJMPZコマンドにはPAUSEされた場合の再実行がコマンドが組み込まれています。この機能を有効にするには、PAUSE (STP_D,n)によって、

対象タスクを停止させます。こうして停止したタスクは、CONTコマンドによりJMPZコマンドが再実行されます

。

CONTコマンド実行後、0.1秒間は、再度PAUSE (STP_D,n)を実行しないでください。

JUMP

書式

JUMP P(arg)
JUMP PL(pln;plm)
JUMP argx,argy,argu,argz

使い方

JUMP P(1)
JUMP PL(0;5)
JUMP X Y U Z

機能

ゲートモーション

解説

JUMP P(n) 点nへ、ゲートモーション移動

JUMP PL(n;m) パレットnのm番目へ、ゲートモーション移動

JUMP X Y U Z 座標点へ、ゲートモーション移動

JUMPは複数の動作が組み合わされた、複合コマンドです。このため、PAUSE、STOP、CONTを実行すると、思わぬところで横移動したり、下降してしまうことがあります。

これを防ぐためJUMPコマンドにはPAUSEされた場合の再実行がコマンドが組み込まれています。この機能を有効にするには、PAUSE (STP_D,n)によって、

対象タスクを停止させます。こうして停止したタスクは、CONTコマンドによりJUMPコマンドが再実行されます

。

CONTコマンド実行後、0.1秒間は、再度PAUSE (STP_D,n)を実行しないでください。

サンプルプログラムは、

1行目で、パレット1の2番目の位置のZ位置が500パルス上へ、ゲートモーション移動

2行目は、パルス出力完了待ち

3行目は、メカチャックをOFF、つまりワークピースを離すという意味になります。

```
JUMP PL(1;PT) AD_P(Z_A,500)
WAIT RR(ALL_A)==0
OFF 14
```

LIMZ

書式

LIMZ arg1 [arg2]

使い方

LIMZ -5000
LIMZ -5000 100

機能

JUMP(ゲートモーション)の高速化

解説

JUMPは、Z軸上昇後、XYU軸を移動させます。

デフォルトでは、Zの値が0になるまで移動(上昇)しますが、装置の速度が遅くなります。

LIMZによって、この上昇天井を定めることができます。

LIMZ -1000の場合、-1000の位置まで上昇します。

arg2は、Z軸の上昇が開始されてarg2 msec経過したら、XYU移動を開始させます。

これにより、ゲートモーションの出発点側の動きがアーチとなります。

MOVL

書式

```
MOVL P(n) [option]  
MOVL PL(n;m) [option]  
MOVL arg1,arg2,arg3,arg4 [option]
```

使い方

```
MOVL P(1)  
MOVL P(1) AD_P(X_A,100)  
MOVL X Y U VOID  
MOVL PL(1;1)  
MOVL P(3) VOID_U
```

機能

指定点あるいは、指定座標への直線補間移動。(座標管理による直線補間パルス発生)

解説

MOVLは座標管理された、補間パルス発生です。

引数には直接の座標値、点データ、パレット点などを与えることができます。

ただし、補間は3軸までしか対応できないため、4軸移動になる場合はエラーになります。

optionにはAD_P,X_A|Y_A,VOID_Uなどの補正関数や軸指定定数を追加できます。

X_A|Y_Aとすると指定軸の補間、VOID_Uなどでは指定軸の無視(パルス発生しない)等ができます。

MOVS

書式

```
MOVS [axis] n  
MOVS arg1 [arg2,arg3,arg4]
```

使い方

```
MOVS x y u z  
MOVS X_A n  
MOVS x VOID u z
```

機能

座標管理をしたパルス発生。

解説

補間を伴わない絶対位置パルス発生。MPC-2000では座標管理をしています。

MOVSでは、現在位置と指定された値の差をとって、差分量のパルスを発生します。

尚、短軸の場合は、軸指定定数で指定できます。また、引数にVOIDを指定すると、その軸は動作しません。

MOVT

書式

MOVT axis Point [CCW|CW|0]

使い方

```
MOVT X_A|Y_A P(101)
MOVT X_A|Y_A P(102) CCW
MOVT X_A|Y_A P(i) M(i)
```

機能

座標値による連続補間移動

解説

点データによる連続補間です。絶対値でデータを入力しますが、実際には、起点(ここではP(100))からの相対座標に変換されての移動となります。

第3パラメータにCCW,CWを入れると円弧補間となります。

第3パラメータが無いか、0をすれば、直線補間となります。

サンプルプログラムは、点データを使った汎用プログラムの例です。

P(1000) ~ に座標データ P(2000) ~ に指令データをセットすると、さまざまな連続移動が可能になります。

```
PG 0
ACCEL 8000
CLRPOS
GOSUB *SET_POINT
axis=X_A|Y_A
MOVL axis P(1000)
WAIT RR(axis)==0
FEED axis Y(2000)
DS_DACL
FOR pnt=1001 TO X(2000)
MOVT axis P(pnt) X(1000+pnt)
```

```
NEXT pnt
EN_DACL
WAIT RR(axis)==0
END
*SET_POINT
SETP 1000 10000 20000 0 0 : 'Start
SETP 1001 30000 20000 20000 20000 : 'Cir target and Center
SETP 1002 30000 10000 0 0
SETP 1003 10000 10000 20000 10000
SETP 1004 10000 20000 0 0
SETP 2000 1004 50 0 0 : 'End of Point and FEED value
SETP 2001 CW 0 0 0 : 'P(1000) to P(1001) CW
SETP 2002 0 0 0 0 : 'P(1001) to P(1002) linear
SETP 2003 CW 0 0 0 : 'P(1002) to P(1003) CW
SETP 2004 0 0 0 0 : 'P(1003) to P(1004) linear
RETURN
```

MPG

書式

MPG arg [taskn]
MPG

使い方

MPG 1
MPG 1 4

機能

MPGボードのアサイン

解説

どのMPGボードを使用するか決定。タスクごとに指定できる。

tasknを指定しなければ、実行されたタスクでMPGを指定。

指定すると、指定タスクが、そのMPGを使用する。指定結果は、MPGでリストできる。

同様のコマンドにPGがあるが、こちらは、指定PGの存在の有無について判定しない。

MPGコマンドでは存在しないIPG番号を指定すると、エラーが表示される。

なお、

MPG 0～9 **が**、MPG-2314となり直線・円弧補間まで対応。

MPC-1200 のJ8コネクタからのパルス発生は MPG 17 です。

同機能コマンドとしてPGがありますが、こちらは、存在しないIPG番号を指定しても、エラーとはなりません。

NEWP

書式

NEWP

使い方

機能

点データ初期化

解説

点データをすべて0に初期化します。

PALLET

書式

PALLET h P(i) P(j) P(k) [P(l)] m n * 0 P(2)の場合は以下のように記述します。12_48以後
PALLET 1 P(1) P(2) m

12_48以前では、以下のように記述してください。

PALLET 1 P(1) P(2) P(2) m 2

2を指定する理由は、0の除算を発生させないためです。

```
'4points teaching
SETP 1 0 0 0 -5000
SETP 2 20000 0 0 -5000
SETP 3 0 20000 0 -5000
SETP 4 20000 20000 0 -5000
PALLET 1 P(1) P(2) P(3) P(4) 4 3
FOR I_=1 TO 12
  JUMP PL(1;I_)
  WAIT RR(ALL_A)==0
NEXT
or
FOR I_=-1 TO -12 STEP -1
  JUMP PL(1;I_)
  WAIT RR(ALL_A)==0
NEXT
-----
'3points teaching
SETP 1 0 0 0 -5000
SETP 2 20000 0 0 -5000
SETP 3 0 20000 0 -5000
PALLET 1 P(1) P(2) P(3) 4 3
FOR I_=1 TO 12
  JUMP PL(1;I_)
  WAIT RR(ALL_A)==0
NEXT
-----
'linear tray
SETP 1 0 0 0 -5000
SETP 2 20000 0 0 -5000
PALLET 1 P(1) P(2) 4
FOR I_=1 TO 4
  JUMP PL(1;I_)
  WAIT RR(ALL_A)==0
NEXT
```

PG

書式

PG arg1 [taskn]
PG

使い方

PG 0
PG 1 2

機能

MPGボードの指定

PG 0 ~ 9 が MPG-2314 高機能 円弧補間まで可

MPC-1200 のJ8コネクタからのパルス発生は PG 17

解説

PG コマンドはMPGコマンドと同機能ですが、(MPGコマンド参照)MPGボードの存在テストをしません。
このために装備していないMPGを指定してもエラー表示しません。

PGA,PGB

書式

PGA str\$ val

使い方

PGA "G" 1000
PGB "V"
pr V_PGB

機能

MPC-1000,MPC-N816のPG制御コマンド

解説

MPC-1000,MPC-N816には、PGAとPGBの2つの簡易PG機能があります。
それぞれのPGを制御するコマンドがPGA,PGBで、書式と機能は以下のとおりです。
ここでは、PGAの例ですが、PGBも同一の書式で使用出来ます。

PGA "G" pps ; PPS指定パルス発生(20 ~ 9000pps)
PGA "S" pps ; パルスレート設定(20 ~ 9000pps)
PGA "W" duty ; PWM(40 ~ 970/1000)
PGA "P" pls ; パルス数指定パルス発生(+/-8000000)
PGA "A" pps ; 加減速テーブル生成(500 ~ 12000pps)
PGA "F" f ; 速度選択(10 ~ 0)
PGA "R" pls ; 加減速パルス発生 相対(+/-8000000)
PGA "M" pos ; 加減速パルス発生 座標(+/-8000000)
PGA "H" pos ; 現在位置設定(+/-8000000)
PGA "D" n ; パルス方式(0:デフォルト2PLS 1:方向指示)
PGA "C" ; 現在位置取得
PGA "V" ; バージョン取得

"PGA C"および"PGA V"コマンド発行後のリターン値は、V_PGAに代入されます。

(PGBの場合は、V_PGB)

なお、パルス発生は、それぞれ、OFF PGA,OFF PGBで停止することができます。

PLIST

書式

PLIST arg1

使い方

PLIST
PLIST 10

機能

点データの表示

解説

点データの連続表示です。20点ずつ表示してタイプイン待ちとなります。
'q'で終了。それ以外のキーでは継続となります。

```
#plist
P(1) X= 200 Y= 0 U= 0 Z= 0
P(2) X= 0 Y= 0 U= 0 Z= 0
P(3) X= 0 Y= 0 U= 0 Z= 0
P(4) X= 0 Y= 0 U= 0 Z= 0
P(5) X= 0 Y= 0 U= 0 Z= 0
P(6) X= 0 Y= 0 U= 0 Z= 0
P(7) X= 0 Y= 0 U= 0 Z= 0
P(8) X= 0 Y= 0 U= 0 Z= 0
P(9) X= 0 Y= 0 U= 0 Z= 0
```

RANGE

書式

RANGE axis pos_limit neg_limit

使い方

```
RANGE X_A 10000 -10000
RANGE X_A|Y_A 20000 0
RANGE X_A|PR_CHK 1000 -1000
RANGE VRING|X_A 1000
```

機能

可能動作領域の設定

解説

RANGEはそれぞれの軸に対して、ソフトリミットとなる限界値を設定します。

RANGEコマンドは内部のレジスタに値を設定するのみです。

この値によるソフトリミットを有効にするには、INSETコマンドの引数にSLMT_ONを追加します。

例:INSET X_A XXXXX|SLMT_ON

axis指定に定数PR_CHKをORするとPtoP制御コマンド(RMVS,MOVL,JUMPなど)に対して移動先座標チェックが行われます。移動先が指定範囲内でない場合は、エラー表示、停止します。移動先チェックは、RMVC,RMVT等のコマンドには無効です。SLMT_ONと併用してください。

axis指定に定数VRINGをORすると、内部の位置カウンタがリングカウンタになります。

リングカウンタとは、回転軸の位置管理などに用います。

VRINGを指定するとソフトリミットは無効です。

30 RANGE Z_A|PR_CHK 1000 -1000
40 INSET X_A|Y_A LMT_ON|SLMT_ON

RMVC

書式

RMVC axis arg

使い方

RMVC X_A CW
RMVC Y_A CCW

機能

量を指定しないパルス発生。CW,CCWは方向指定。+1,-1でも可。

解説

argで正の数を指定するとCW方向、負の数を指定するとCCW方向のパルス発生となります。

RMVL

書式

RMVL arg1 [arg2,arg3,arg4]

使い方

RMVL x y 0 0

RMVL x y 0 z

RMVL 0 y u z

機能

直線補間でパルス発生する。

解説

3軸までの直線補間でパルス発生をします。4軸指定するとエラーになります。

補間での速度は、X>Y>Z>Uの順序で有効軸の速度が使われます。

RMVL 0 y u z

であれば、yuzの直線補間となり、速度はy軸の速度が使われます。

RMVL X_A|Y_A 10000 というような軸指定の記述はできません。

RMVS

書式

RMVS [axis] n
RMVS X [Y,U,Z]

使い方

RMVS X_A n
RMVS x y u z

機能

指定量のパルスを発生します。

解説

加減速付きの相対パルス発生命令です。正の値でCW方向。負の値でCCW方向です。

RMVS X_A|Y_A 10000 というような軸指定の記述はできません。

RMVT

書式

RMVT axs arg1 arg2 [CCW|CW]0 cent1 cent2]

使い方

```
RMVT X_A|Z_A 20000 0
RMVT X_A|Z_A 0 20000 CCW 0 10000
```

機能

連続補間移動

解説

相対座標による連続補間コマンドです。第3パラメータにCCWかCWを与えると円弧補間となります。

この場合は、円弧中心を指定するパラメータが必要となります。

いずれの座標値もコマンドが実行される場所からの相対座標となります。

第3パラメータが無いか、0とすれば、直線補間となります。

例では、X,Yの円弧補間を実施します。

EN_DACK,DS_DACLは、減速の有効・無効です。

図はRMVT X_A|Y_A 0 20000 CCW 0 10000の実行イメージです。

```
PG 0
ACCEL 8000
CLRPOS
DS_DACL
RMVT X_A|Y_A 20000 0
RMVT X_A|Y_A 0 20000 CCW 0 10000
RMVT X_A|Y_A 0 -20000 CCW 0 -10000
RMVT X_A|Y_A 10000 0
EN_DACL
```


SET

書式

SET n x y u z

使い方

SET 0 1 1 1 1

SET 1 5 5 5 1

機能

TEACHコマンドでのイン칭ング量の設定

解説

TEACHコマンドではxyuzキーでイン칭ングすることができますが、その量を設定します。

SETで指定できるエリアは4つあり、0~3を指定してそれぞれの値を設定します。

TEACHコマンドで相当する'0'~'3'のキーを押すと、設定されたイン칭ング量が呼び出されます。

SETP

書式

```
SETP n arg1 arg2 arg3 arg4
SETP n P(m)
SETP n PL(m;l)
SETP n strng
```

使い方

```
SETP 1 100 100 20 3
SETP 2 X(0) Y(0) U(0) z(0)
SETP 13 P(3)
SETP 100 "abcdef"
```

機能

点データに値を設定する。

nに0を指定すると現在位置。

nに-1を指定するとエンコーダ・カウンタ、引数に文字列を使用すると、文字列収納。

解説

点データの編集コマンドです。引数にP(n)

PL(m;n)を指定することができるので、点データのコピー生成にも使用できます。

また、点データエリアには、文字列を収納することもできます。文字列引数は単項のみです。(a\$,"
",str\$())などで'+結合は不可)

```
FOR i=1 TO 10
  SETP i STR$(i-5)
NEXT
FOR i=1 TO 10
  PRINT P$(i)
NEXT
FOR i=1 TO 10
  a$=STR$(i-5)+" Volt"
```

SETP i a\$
NEXT

SET_MCX

書式

SET_MCX axs Cmd WR6+WR7

使い方

SET_MCX Z_A &h0006 400

機能

MCX314コマンド直接設定

解説

SYNCコマンドで、動作トリガを規定し、そのトリガにより一定量パルス発生する場合は、コマンドとパルス値を直接MCX-314にSET_MCXコマンドにより設定します。
サンプルでは、X軸のカウンタが100になったら、Z軸を50パルス出力することになっています。
SET_MCX Z_A &h0006 50
は、Z軸に対するコマンドで、移動量を指定するコマンド06と移動量50を指定しています。
コマンドの仕様については、MCX-314のデータシートを参照してください。

```
ACCEL Z_A|OUTSL 1000000 10000 1000000
ACCEL X_A|OUTSL 3000
INSET X_A CMP_CNT|PHASE2
```

```
SYNC X_A &H00004001 0
SET_MCX Z_A &h0006 50
SYNC Z_A 0 1
CLRPOS Z_A
RANGE X_A 100 0
```

```
WAIT CMP_C(Z_A)!=0
WAIT RR(Z_A)==0
```

SHOM[MPC-1200]

書式

SHOM pat

使い方

SHOM &H50
SHOM &H55

機能

HOMEコマンドでSD入力の無効化

解説

原点復帰実行時でのニアオリジンのスローダウントリガ、SD入力の有効、無効を設定します。

Bitの割り付けは以下のとおりです。

BIT0:SDX, BIT2: SDY,BIT4 :SDU,BIT6:SDZ

BITに1を立てると無効。0にすると有効です。

```
SHOM &H55 ' ALL SDinput ignored
HOME ALL_A NEG_L
WAIT RR(ALL_A)==0
CLRPOS ALL_A
```

SHOM[MPG-2314]

書式

SHOM axis patn
SHOM patx paty patu patz

使い方

```
SHOM X_A|Z_A|Y_A IN0_ON|IN1_OFF  
SHOM X_A|Z_A|Y_A IN0_ON|IN1_OFF|CW  
SHOM X_A|Z_A|Y_A IN0_ON  
SHOM IN0_ON 0 0 0
```

機能

原点復帰の条件を決定する。

解説

MPG-2314の原点復帰検出センサは各軸2つずつありIN0とIN1に区別されています。
例えばY軸の場合は、MPG-2314のJ4上では、YIN0,YIN1と名づけられています。

IN0はニアオリジン IN1はZ相を想定していますので必要に応じて設定します。
また、SHOMの設定は、HOMEコマンドを実行しない限り有効ではありません。

SHOM X_A|Z_A|Y_A IN0_ON

この場合はニアオリジンのみの原点復帰を想定します。ニアオリジンをオン検出すると停止します。

SHOM X_A|Z_A|Y_A IN0_ON|IN1_OFF|CW

この場合はX,Y,Z軸に対して動作を規定しています。

ニアオリジン停止後、Z相サーチとなります。サーチ方向はCW方向です。

ニアオリジンはON検出、Z総は、OFF検出です。CW/CCWを省略するとCCW方向となります。

SPEED

書式

SPEED [axs] n

使い方

SPEED n

SPEED X_A n

機能

パルス発生のpps設定

解説

パルス発生をACCELで指定した最高速度pps以下で、n pps指定できる。

FEEDコマンドより、速度を細かくドライブ速度を指定できる。ただし分解能は、(最高速度/8192) ppsとなる。

サンプルプログラムのように、パルス発生中の細かな速度変更にも有効。

```
40 ACCEL 40000 1000
50 RMVC U_A 1
60 DO
70 FOR i=1 TO 10
80 SPEED U_A i*4000
90 TIME 100
100 NEXT
110 FOR i=10 TO 1 STEP -1
120 SPEED U_A i*4000
130 TIME 100
140 NEXT
150 LOOP
```

STOP

書式

STOP axis arg1

使い方

STOP X_A STP_D
STOP ALL_A IN1_ON
STOP X_A|Y_A VOID

機能

停止命令発行もしくは停止モードの設定

解説

STOP X_A STP_D

このタイプのコマンドは、対象MPGに対して減速停止、あるいは即停止命令を発行します。STP_Dは減速、STP_Iは即停止です。

プログラム例は、動作中に入力スイッチで停止させる方法の例です。

STOP ALL_A IN0_ON|IN1_OFF

このタイプのコマンドは、MPG-2314の入力ポートの機能を決めます。

IN0_ON|IN1_OFF の場合は、IN0(S0)がオン、IN1(S1)がオフになると停止します。

停止条件は、コマンド実行後維持されますので、解除する場合は引数にVOIDを与えます。

INnによる停止は、ACCELコマンドでドライブ速度>初速度の場合は、減速停止ですが、ドライブ速度==初速度とすると即停止となります。

停止条件の解除はVOIDを指定します。

STOP X_A VOID

EX1:

MOVL 10000 10000 0

WHILE RR(ALL_A) : IF SW(192) THEN : STOP STP_D : END_IF : WEND

EX2:

STOP X_A IN0_OFF /* setting the STOP condition

RMVS X_A POS_L /* Generating pulse

WAIT RR(X_A)==0

STOP X_A VOID /* clear the STOP condition

RMVS X_A 1000

STPS

書式

STPS axis n
STPS argx [argy,argu,argz]

使い方

STPS X_A 1000
STPS 100 200 300 400
STPS VOID 100 200
STPS X_A|Y_A 1000

機能

現在位置設定

解説

軸指定した場合は、相当軸に同じ値を設定します。
引数を並べた場合は、X Y U Z の順序で設定できます。
VOIDが与えられているか、省略された引数の軸は設定されません。

SYNC

書式

SYNC axs WR6 WR7

使い方

SYNC X_A &H00004001 0
SYNC Z_A 0 4

機能

MCX-314レジスタ設定

解説

MPG-2314に搭載されているMCX-314には、ハード的にリアルタイム処理を行う機能があります。

X軸が一定パルスを超えたら、他の軸を起動する。

入力信号がはいったら、その時点のカウンタ値をラッチする

このコマンドにより、このような機能をハード上の機構によりリアルタイムに実行することができます。

WR6,WR7に実際どのような値をセットするかは、MC-314のデータシートを参照してください。

サンプルプログラムでは、X軸のカウント値(100)により、Z軸を起動し、その値が50pulseになったところで、出力ポート(O3)をオンするというものです。

```
ACCEL Z_A|OUTSL 1000000 10000 1000000  
ACCEL X_A|OUTSL 3000  
INSET X_A CMP_CNT|PHASE2
```

```
SYNC X_A &H00004001 0  
SYNC Z_A 0 4  
CLRPOS Z_A  
RANGE Z_A 50 0  
RANGE X_A 100 0
```

```
WAIT CMP_C(Z_A)!=0  
STOP Z_A STP_I  
WAIT RR(Z_A)==0
```

TEACH

書式

TEACH

使い方

TEACH
T

機能

インチング操作による点データの教示

解説

TEACHコマンド実行前に、PGが選択され、ACCELコマンドが実行されていなければなりません。

TEACHコマンドを実行すると、以下のように現在位置とインチング量が表示されます。

それぞれの軸は以下のキーでインチングします。

x,X

y,Y

u,U

z,Z

#t

PG=[1] X=1200 Y=0 U=0 Z=0 dx=200 dy=200 du=200 dz=200 P(30

Pコマンドを押すと点番号入力待ちとなります。番号を入力すると指定された点データに現在地がセットされます

。

0～3のキーでSETコマンド設定されたインチング量を選択することができます。

なお、対象PGがMPG-2314の場合、位置表示とあわせてエラー表示もします。

PG=[1] X=1200 Y=0! U=0 Z=0 dx=200 dy=200 du=200 dz=200

エラーとなった軸の数値の後ろに!が表示されます。

WARP

書式

WARP [AXS] [up_z] P(n) [dwn_z] [HAND p1 p2 logic] [WHEN logic]

使い方

```
WARP 1000 P(100) 500 HAND 1 VOID X(0)>2000 WHEN SW(192)==1
WARP X_A|Y_A 0 PL(1;1_) 1000 HAND OFF|15 OFF|2 HALF_P WHEN SW(193)==0
WARP X_A|U_A 1000 P(100) 500
```

機能

高速ゲートモーション

解説

アーチ状のゲートモーションです。

AXSを指定する場合は、Z以外の不要な軸を外して動作させることができます。

up_zを指定すると、パルス分上昇してから、XY移動を開始します。

dwn_zを指定すると、下降において最下点よりdwn_zパルス分直線下降を確保します。

HAND文を追加すると、条件が成立した場合に、ON/OFF操作を行うことができます。

ON/OFF いずれかを無効にしたい場合は、引数にVOIDを与えます。

また、条件に、定数HALF_Pを指定すると、XY移動の midpoint でON/OFF操作を行います。

Z下降が条件成立前に開始される場合は、Z下降直前にON/OFF制御を実施します。

WHEN文を追加すると、条件が成立した場合のみ下降します。

```
PG 0
ACCEL 10000
CLRPOS
ON 2
MOVS P(1) : WAIT RR(ALL_A)==0
WARP 100 P(5) HAND 1 2 X(0)>2000 WHEN SW(5)==1
```

YPLS

書式

YPLS Var1 Var2 Var3 [Count]

使い方

YPLS Port Rate Counter
YPLS Port Rate Counter 1000

機能

10pps ~ 5000ppsのI/O パルス発生

解説

出力ポートによるパルス発生です。タスクとは無関係に、変数操作で、停止、速度の変更、実行中のパルス数管理ができます。

Port,Rate,Counterは変数で直接数値を指定できません。変数に値を代入してからコマンド実行します。

Port 768 , 2--> 769

その他では、 1--> 12 , 2--> 13

Rate

ZPLS

書式

ZPLS Var1 Var2 Var3 [Count]

使い方

ZPLS Port Rate Counter
ZPLS Port Rate Counter 1000

機能

10pps ~ 5000ppsのI/O パルス発生

解説

出力ポートによるパルス発生です。タスクとは無関係に、変数操作で、停止、速度の変更、実行中のパルス数管理ができます。

Port,Rate,Counterは変数で直接数値を指定できません。変数に値を代入してからコマンド実行します。

Port 770 , 2--> 771

その他では、 1--> 14 , 2--> 15

Rate

AD_P

書式

AD_P(axes,n) n= \pm 32767
AD_P(P(n))

使い方

```
MOVS P(n) AD_P(X_A,1000)
MOVS P(n) AD_P(X_A,1000) AD_P(Z_A,-1000)
JUMP P(n) AD_P(P(m))
```

機能

移動点補正

解説

MOVSなどの点データ引数(座標値)に補正値を加える。指定点の上で停止させるのに使用します。

また画像処理でX,Y点などを一時的に補正させるのに使用できます。

点データを指定すると、四軸の座標値がそのまま加算されます。

この演算では、点データそのものを修正しません。AD_P(axes,n)の場合の補正範囲は \pm 32767の範囲です。

```
MOVS P(5) AD_P(X_A,1000) => MOVS X(5)+1000 Y(5) U(5) Z(5)
MOVS P(6) AD_P(X_A,1000) AD_P(Z_A,-1000) => MOVS X(6)+1000 Y(6) U(6) Z(6)-1000
```

CMP_C

書式

CMP_C(axis)
CMP_C(port,axis)

使い方

```
WAIT CMP_C(X_A)==2  
A=CMP_C(16,X_C)
```

機能

カウンタとCOMP+/-の比較結果を参照する。
カウンタとCOMP+/-の比較結果が変化したら、指定ポートをONする。

解説

MPG-2314には、COMP+,COMP-レジスタがあり、カウンタとCOMPレジスタの比較をリアルタイムで行うことができます。比較結果は、CMP_C関数で参照します。

CMP_C() = [BIT0 = COMP-レジスタ

又、比較カウンタ値には、現在位置カウンタ、エンコーダカウンタの何れかを選ぶ事ができます。
INSET X_A CMP_PLS と設定すると、パルス位置とCOMPレジスタの比較結果をCMP_C(X_A)で知る事ができます。
INSET X_A CMP_CNT の場合は、エンコーダカウンタとの比較になります。

COMP レジスタはRANGE コマンドで設定する事ができます。

RANGE X_A COMP+ COMP-

なお、サンプルプログラムのように、CMP_C(port,X_A)

記述すると、比較フラグの変化を待って指定ポートをONして抜け出します。
この場合、CMP+,CMP-のいずれのビットが変化しても変化検出となります。

```
40 ACCEL 30000
50 CLRPOS
60 INSET CMP_PLS
65 P_DET=500
70 RANGE X_A P_DET P_DET
80 RMVC X_A 1
100 DO
110 A=CMP_C(16,X_A)
120 INC P_DET 500
130 OFF 16
135 RANGE X_A P_DET P_DET
140 LOOP
```

CMP_P

書式

CMP_P([axs,],v)

使い方

CMP_P(n)
CMP_P(axs,n)

機能

現在位置と点データの比較

解説

現在位置と指定された点データを比較します。軸指定がなければ、XYZUの全軸の値を比較し、同じであれば1、1軸でも相違していれば、0を返します。

軸指定を与えると、指定した軸のみ比較します。

```
10 PG 0
15 CLRPOS
20 ACCEL 8000
30 SETP 7000 10000 20000 30000 40000
40 MOVS P(7000)
50 DO
60 IF CMP_P(7000) THEN : PRINT "Arrived" : BREAK : END_IF
70 TIME 100
80 LOOP
90 RMVS Z_A 100
100 WAIT RR(Z_A)==0
110 PRINT CMP_P(7000)
120 PRINT CMP_P(VOID_Z,7000)
#run
Arrived
0
1
```

#

HPT

書式

HPT(arg1)

使い方

```
prx HPT(0)
WAIT HPT(XIN0)==1
IF HPT(XIN0)==0 : RMVS X_A 20000 : END_IF
```

機能

PGボードの原点復帰入力ポートを読み出す。

解説

【MPG-2314】

HPT(0) パラレルでIN0～IN3,INPOS,ALM入力を読み取る。

パラレル時は8bit単位各軸で32bitパラ出力となります。

1) HPT(0)={U}{Z}{Y}{X}

{8bit}=ALM(bit7),INP(bit6),IN3(bit3)IN1(bit2,bit1)IN0(bit0)

IN1(bit2,bit1)は、MPG-2314では、IN1入力が、内部でIN1,IN2とショートされていることを示しています。

このため、IN1をオンとするとIN2もオンとなり、この場合"hpt(0) -> 00000006" となります。

2) HPT(XIN0) 指定された、ポートを読み取る。

原点復帰入力: XIN0,XIN1,XIN2,XIN3～UIN0,UIN1,UIN2,UIN3

ALM入力: XALM～UALM

INPOS入力: XINP～UINP

HPTで読み取られる各指定ポートがオンであれば1となります。

LMT

書式

LMT(n)

使い方

```
IF LMT(X_A,LMTp)!=0 THEN  
  RMVS X_A -10000  
END_IF
```

機能

エラー入力読み取り

解説

【MPG-2314】

LMT(0)とすると、XYZUすべての軸のエラー状態を参照できます。

byte= { EMG,ALM,LMTn,LMTp,SLMTn,SLMTp}

UbyteZbyteYbyteXbyte

PGE

書式

PGE(0)
PGE(axes,val)

使い方

```
IF PGE(X_A,ALM) THEN : GOTO *EMG_X_A : END_IF  
IF PGE(0) THEN : GOTO *EMG : END_IF  
IF PGE(X_A,CLR_ER|ALM) THEN : GOTO *EMG_X_A : END_IF  
IF PGE(CLR_ER) THEN : GOTO *EMG : END_IF
```

機能

MPG-2314の停止原因の参照

解説

MPG-2314はEMG,ALM,LMT,IN0～IN1の各入力によってパルス発生を停止させることができます。

停止後、原因入力解除されても、PGE()関数で、停止原因を知ることができます。

引数の指定にはふたとおりあり、

PGE(0)の場合、4軸すべての停止原因フラグを参照できます。

この場合、PGE(0) = {Uaxs|Zaxs|Yaxs|Xaxs} で 4byte構成となります。

各byteのビット構成は、{EMG,ALM,LMTn,LMTp|IN3,IN2,IN1,IN0}の8bitです。

もうひとつの方法は、軸指定とビット指定定数によりエラーを個別にチェックする方法です。

PGE(X_A,LMTp) LMTpをテスト

PGE(X_A,(IN1|LMTp)) LMTp、IN1の双方をテスト

ビット指定を0とすると、指定軸のエラー情報をbyteで返します。

なお、引数にCLR_ERのみをセットすると、全ステータスの取得と同時にエラーステータスをクリアします。

また、軸指定の場合、ビット条件にCLR_ERをORしておくと、該当軸のみリード&クリアとなります。

```
LIST
10 'XXXX=CLR_ER
20 'XXXX=(X_A,CLR_ER|IN0)
50 PG 1
60 ACCEL 4000
70 STOP ALL_A IN0_ON
80 OFF 0 1
90 CLRPOS
100 MOVS 1000000 1000000 100000 100000
110 TIME 1000
120 ON 0 1
130 WAIT RR(X_A)==0
140 PRX PGE(0)
150 PRX PGE(XXXX)
160 PRX PGE(0)
#run

00000101
00000001
00000100
#prx XXXX
0001F100
#
```

PL

書式

PL(n;m)

使い方

MOVS PL(1;10)
JUMP PL(2;100)

機能

パレット点を計算してMOVS等の移動コマンドでその点データを引き渡す。

解説

PALLET 1 P(1) P(2) P(3) P(4) 4 3

JUMP PL(1;l)

パレットコマンドを実行した後で用います。パレットは0～63指定できます。
パレット番号と、パレット点の区切りは、";"であることに留意してください。
";"で区切るとパレットを正しく選択できません。

引数を負にするとZIG_ZAG順になります

RR

書式

RR(arg1)

使い方

```
WAIT RR(X_A)==0  
WAIT RR(ALL_A)==0  
IF RR(X_E)!=0 THEN
```

機能

MPGの動作状態の監視

解説

RR(arg1)

は、ステータスとarg1でANDをとった値を返します。(arg1 & ステータス)

ただし、arg1が0の場合はANDをしないで、動作ステータスをそのままよみとって返します。

通常は、WAIT RR(X_A|Y_A)==1

のように動作軸の停止を監視します。

MPG-2314に対するステータス読み取りでは、上位4bitが各軸のエラー状態となります。

X_E ~ U_E, ALL_Eという予約定数が対応します。

*ステータスとはMCX-314AsのRR0レジスタ

RR3

書式

RR3(axis)

使い方

A=RR3(X_A)

機能

MPG-2314の割り込みフラグ読み取りおよびフラグ解除

解説

MPG-2314で割り込みを設定すると、割り込み発生レジスタRR3の内容に従って、INT割り込みが発生します。割り込みの解除は、割り込み条件の解除と、このRR3レジスタの読み取りが必要になります。

RR3レジスタは各軸に独立に用意されているため、その読み出しには軸指定が必要です。

このため、RR3(ALL_A)というような記述はできません。

必ず、RR3(X_A),RR3(U_A) というように、単1軸を指定して読み出します。

得られた値(1byte)の意味は下記のとおりです。

bit7:D_END bit6:C_STA bit5:C_END bit4:P>=COMP+ bit3:P

XYZU

書式

X(arg1)
Y(arg1)
U(arg1)
Z(arg1)

使い方

```
MOVS X(1)+A VOID U(1)+B VOID  
setp 1 x(0) y(0) u(0) z(0)
```

機能

現在地、および点データの座標を返す。

解説

arg1が0の時、現在位置を返します。

0以外の数値では、指定された番号の点の座標値を返します。

PG_TASK0

書式

PG_TASK0

使い方

```
print PG_TASK0
```

機能

PG番号取得

解説

タスク0に割り当てられているPG番号を返す変数。
存在しないPGの場合は-1となります。

```
/* USE MPG-2314 #0  
10 PG 0  
20 PRINT PG_TASK0  
30 PG 10  
40 PRINT PG_TASK0  
50 PG 1  
60 PRINT PG_TASK0  
#run
```

```
0  
10  
-1
```

ALL_A

書式

ALL_A

使い方

FEED ALL_A 100

機能

全軸指定

解説

対象ボード: MPG-2314/MPC-1200

ACCEL ALL_A 30000 1000 500 /* Acceleration/deceleration setting
FEED ALL_A 100 /* Speed setting
INSET ALL_A MD_2PLS|ALM_OFF|LMT_OFF /* In port set
STOP ALL_A STP_D /* Moving stop with deceleration
WAIT RR(ALL_A)==0 /* Wait until moving complete
etc

ALL_E

書式

ALL_E

使い方

RR(ALL_E)

機能

全軸エラー指定

解説

対象ボード: MPG-2314

移動後のエラーの有無を調べます。

次のビットのどれかが立ったことを表します。

RR1レジスタ(ドライブ終了ステータス) ENG,ALARM,LMT-,LMT+

RR2レジスタ(エラー情報) EMG,ALARM,HLMT-,HLMT+,SLMT-,SLMT+

```
100 MOVL P(1)
110 WAIT RR(ALL_A)=0
120 IF RR(ALL_E)!=0 THEN /* Confirming error status
130 PRINT "ERROR STOP"
140 ELSE
150 PRINT "NORMAL STOP"
160 END_IF
170 PRX RR(ALL_E)
```

ALM

書式

ALM

使い方

LMT(X_A,ALM)

機能

エラービット指定

解説

対象ボード: MPG-2314

アラーム信号ビット

IF LMT(X_A,ALM)≠0 THEN /* confirming reason for stop

ALM_OFF

書式

ALM_OFF

使い方

INSET X_A ALM_OFF

機能

アラーム設定

解説

対象ボード: MPG-2314

アラームOFFで有効

INSET X_A ALM_OFF /* X-axis 'ALARM' enabled on signal 'OFF'

ALM_ON

書式

ALM_ON

使い方

INSET X_A ALM_ON

機能

アラーム設定

解説

対象ボード: MPG-2314

アラームONで有効

INSET X_A ALM_ON /* X-axis 'ALARM' enabled on signal 'ON'

CCW

書式

CCW

使い方

RMVC X_A CCW

機能

原点復帰サーチ方向指定

円弧補間指定

解説

SHOMでは原点復帰のZ相サーチ方向を指定します。

MOVTでは円弧補間の回転方向を指定します。

SHOM X_A|Y_A IN0_ON|CCW /* set HOME condition. CCW movement until the sensor turns on
MOVT X_A|Y_A P(102) CCW /* continuous interpolation. CCW revolution.
RMVC X_A CCW /* infinite pulse generation. CCW movement.

CLR_ER

書式

CLR_ER

使い方

PGE(CLR_ER)
PGE(Z_A,CLR_ER|LMTp)

機能

エラーステータスクリア

解説

PGE()参照

CMP_CNT

書式

CMP_CNT

使い方

INSET X_A CMP_CNT|PHASE1

機能

カウンタ比較

解説

対象ボード: MPG-2314

エンコーダカウンタとCOMP+と比較

INTA_ONも参照してください。

INSET X_A CMP_CNT|PHASE1

CMP_PLS

書式

CMP_PLS

使い方

INSET X_A CMP_PLS

機能

カウンタ比較

解説

対象ボード: MPG-2314

現在パルスカウンタとCOMP+と比較

INTA_ONも参照してください。

INSET X_A CMP_PLS

CW

書式

CW

使い方

RMVC X_A CW

機能

原点復帰サーチ方向指定

円弧補間指定

解説

対象ボード: MPG-2314

SHOMでは原点復帰のZ相サーチ方向を指定します。

MOVTでは円弧補間の回転方向を指定します。

SHOM X_A|Y_A IN0_ON|CW /* set HOME condition. CW movement until the sensor turns on
MOVT X_A|Y_A P(102) CW /* continuous interpolation. CW revolution.
RMVC X_A CW /* infinite pulse generation. CW movement.

C_LESS

書式

C_LESS

使い方

STOP X_A C_LESS

機能

カウンタ比較

解説

対象ボード: MPG-2314

カウンタ <COMP+で割り込み

INTA_ON,INTB_ON参照

see also INTA_ON

C_MORE

書式

C_MORE

使い方

STOP X_A C_MORE

機能

カウンタ比較

解説

対象ボード: MPG-2314

カウンタ>=COMP+で割り込み

INTA_ON,INTB_ON参照

see also INTA_ON

EMG

書式

EMG

使い方

LMT(X_A,EMG)

機能

エラービット指定

解説

対象ボード: MPG-2314

緊急停止信号(EMGN)ビット

```
IF LMT(X_A,EMG)!=0 THEN /* confirming reason for stop
```

IN0_OFF

書式

IN0_OFF

使い方

SHOM X_A IN0_OFF

機能

停止入力設定

解説

対象ボード: MPG-2314

XIN0 ~ ZIN0をOFFで有効にします。

SHOM X_A IN0_OFF

STOP X_A IN0_OFF

see also IN0_ON

IN0_ON

書式

IN0_ON

使い方

SHOM X_A|Y_A IN0_ON

機能

停止入力設定

解説

対象ボード: MPG-2314

XIN0 ~ ZIN0をONで有効にします。

```
100 SHOM X_A|Y_A IN0_ON /* set HOME condition.
110 HOME -100000 -100000 0 0
120 WAIT RR(ALL_A)==0
```

```
100 STOP X_A IN0_ON /* set stop condition. if XIN0 turn on then stop.
110 MOVL 5000 0 0 0
120 WAIT RR(X_A)=0 /* wait for stop
130 IF HPT(XIN0)==1 THEN /* confirming reason for stop
140 PRINT "IN0 stop"
150 ELSE
160 PRINT "normal stop"
170 END_IF
```

IN1_OFF

書式

IN1_OFF

使い方

SHOM X_A IN1_OFF

機能

停止入力設定

解説

対象ボード: MPG-2314

XIN1 ~ ZIN1をOFFで有効にします。

SHOM X_A IN1_OFF

STOP X_A IN1_OFF

see also IN0_ON

IN1_ON

書式

IN1_ON

使い方

SHOM X_A IN1_ON

機能

停止入力設定

解説

対象ボード: MPG-2314

XIN1 ~ ZIN1をONで有効にします。

SHOM X_A IN1_ON

STOP X_A IN1_ON

see also IN0_ON

IN2_OFF

書式

IN2_OFF

使い方

STOP X_A IN2_OFF

機能

停止入力設定

解説

対象ボード: MPG-2314

XIN2 ~ ZIN2をOFFで有効にします。

STOP X_A IN2_OFF

see also IN0_ON

IN2_ON

書式

IN2_ON

使い方

STOP X_A IN2_ON

機能

停止入力設定

解説

対象ボード: MPG-2314

XIN2 ~ ZIN2をONで有効にします。

STOP X_A IN2_ON

see also IN0_ON

IN3_OFF

書式

IN3_OFF

使い方

STOP X_A IN3_OFF

機能

停止入力設定

解説

対象ボード: MPG-2314

XIN3 ~ ZIN3をOFFで有効にします。

STOP X_A IN3_OFF

see also IN0_ON

IN3_ON

書式

IN3_ON

使い方

STOP X_A IN3_ON

機能

停止入力設定

解説

対象ボード: MPG-2314

XIN3 ~ ZIN3ONで有効にします。

STOP X_A IN3_ON

see also IN0_ON

INP_OFF

書式

INP_OFF

使い方

INSET X_A INP_OFF

機能

インポジション設定

解説

対象ボード: MPG-2314

インポジションOFFで有効

INP_ON/INP_OFFはどちらかを設定すれば有効、しなければ無効です。

INSET X_A INP_OFF /* X-axis 'INPOSITION' enabled on signal 'OFF'

INP_ON

書式

INP_ON

使い方

INSET X_A INP_ON

機能

インポジション設定

解説

対象ボード: MPG-2314

インポジションONで有効

INP_ON/INP_OFFはどちらかを設定すれば有効、しなければ無効です。

INSET X_A INP_ON /* X-axis 'INPOSITION' enabled on signal 'ON'

LMTn

書式

LMTn

使い方

LMT(X_A,LMTn)

機能

エラービット指定

解説

対象ボード: MPG-2314

ハードリミット-ビット

IF LMT(X_A,LMTn)!=0 THEN /* confirming reason for stop

LMTp

書式

LMTp

使い方

LMT(X_A,LMTp)

機能

エラービット指定

解説

対象ボード: MPG-2314
ハードリミット+ビット

```
IF LMT(X_A,LMTp)!=0 THEN /* confirming reason for stop
```


LMT_OFF

書式

LMT_OFF

使い方

INSET ALL_A LMT_OFF

機能

リミット入力設定

解説

対象ボード: MPG-2314

X-LMT ~ Z-LMT OFFで有効。

リミット検出時は即停止。

入力を無効にすることはできません。

INSET ALL_A LMT_OFF /* 'LIMIT' enabled on signal 'OFF'

LMT_ON

書式

LMT_ON

使い方

INSET ALL_A LMT_ON

機能

リミット入力設定

解説

対象ボード: MPG-2314

X-LMT ~ Z-LMT ONで有効。

リミット検出時は即停止。

入力を無効にすることはできません。

INSET ALL_A LMT_ON /* 'LIMIT' enabled on signal 'ON'

MD_2PLS

書式

MD_2PLS

使い方

INSET ALL_A MD_2PLS

機能

パルス出力方法設定

解説

対象ボード: MPG-2314/MPC-1200

2パルス方式(CW/CCW)にする。

パワーオン後のデフォルトです。通常実行の必要はありません。

INSET ALL_A MD_2PLS /* Set the pulse generator to '2 PULSE' mode

MD_DPLS

書式

MD_DPLS

使い方

INSET ALL_A MD_DPLS

機能

パルス出力方法設定

解説

対象ボード: MPG-2314/MPC-1200

1パルス方式(方向指示)にする。

INSET ALL_A MD_DPLS /* Set the pulse generator to 'DIR/PULSE' mode

NEG_L

書式

NEG_L

使い方

HOME NEG_L NEG_L NEG_L NEG_L

機能

負の大数

解説

負の大数

原点復帰ニアオリジンの移動量を大きくとりたい場合は、POS_L,NEG_Lを用いてください。

これらは、正・負の3byte長の最大数です。

#prx POS_L

007FFFF0

#prx NEG_L

FF80000F

HOME NEG_L NEG_L NEG_L NEG_L

NO_PHASE

書式

NO_PHASE

使い方

INSET NO_PHASE

機能

カウンタ入力設定

解説

対象ボード: MPG-2314

無効

INSET NO_PHASE /* Counter disable

PHASE1

書式

PHASE1

使い方

INSET PHASE1

機能

カウンタ入力設定

解説

対象ボード: MPG-2314

カウンターをエンコーダ入力モードとし、カウント倍率は無しとする。

INSET PHASE1 /* multiplier: 1 time

PHASE2

書式

PHASE2

使い方

INSET PHASE2

機能

カウンタ入力設定

解説

対象ボード: MPG-2314

カウンタをエンコーダ入力モードとし、カウント倍率を2通倍とする。

INSET PHASE2 /* multiplier: twice

PHASE4

書式

PHASE4

使い方

INSET PHASE4

機能

カウンタ入力設定

解説

対象ボード: MPG-2314

カウンタをエンコーダ入力モードとし、カウント倍率を4通倍とする。

INSET PHASE4 /* multiplier: 4 times

POS_L

書式

POS_L

使い方

HOME POS_L POS_L POS_L POS_L

機能

正の大数

解説

対象ボード: MPG-2314

原点復帰ニアオリジンの移動量を大きくとりたい場合は、POS_L,NEG_Lを用いてください。

これらは、正・負の3byte長の最大数です。

#prx POS_L

007FFFF0

#prx NEG_L

FF80000F

HOME POS_L POS_L POS_L POS_L

PR_CHK

書式

PR_CHK

使い方

RANGE PR_CHK|X_A 10000 -10000

機能

移動先チェック

解説

対象ボード: MPG-2314

PR_CHK指定をしておくと、リミット値を超えるかどうか、事前に判断され、超える場合は動作以前にエラー停止となります。

PR_CHKの無いソフトリミット指定は、リミットを越えた時点でスローダウン停止となるため、減速距離分のオーバーシュートが発生します。

RANGE PR_CHK|X_A 10000 -10000

RANGE PR_CHK|Y_A 11000 -10000

RANGE PR_CHK|Z_A 12000 -10000

SLMTn

書式

SLMTn

使い方

LMT(X_A,SLMTn)

機能

エラービット指定

解説

対象ボード: MPG-2314

ソフトリミット-ビット

IF LMT(X_A,SLMTn)!=0 THEN /* confirming reason for stop

SLMTp

書式

SLMTp

使い方

LMT(X_A,SLMTp)

機能

エラービット指定

解説

対象ボード: MPG-2314
ソフトリミット+ビット

IF LMT(X_A,SLMTp)!=0 THEN /* confirming reason for stop

SLMT_OFF

書式

SLMT_OFF

使い方

INSET X_A|Y_A SLMT_OFF

機能

ソフトリミット設定

解説

対象ボード: MPG-2314

ソフトリミットを無効にします。

INSET X_A|Y_A SLMT_OFF /* 'SOFT LIMIT' disable

SLMT_ON

書式

SLMT_ON

使い方

INSET X_A|Y_A SLMT_ON

機能

ソフトリミット設定

解説

対象ボード: MPG-2314

ソフトリミットを有効にします。

10 PG 1

20 RANGE X_A|Y_A 200000 -1000 /* XY axes operative restriction set

30 INSET X_A|Y_A SLMT_ON /* 'SOFT LIMIT' enabled

STP_D

書式

STP_D

使い方

STOP X_A STP_D

機能

停止方法選択

解説

対象ボード: MPG-2314/MPC-1200

減速停止

STOP X_A STP_D /* X-axis Stop with deceleration
STOP ALL_A STP_D /* All-axes Stop with deceleration

STP_I

書式

STP_I

使い方

STOP X_A STP_I

機能

停止方法選択

解説

対象ボード: MPG-2314/MPC-1200

即停止

STOP X_A STP_I /* X-axis Stop without deceleration
STOP ALL_A STP_I /* All-axes Stop without deceleration

UIN0

書式

UIN0

使い方

HPT(UIN0)

機能

HPT入力指定

解説

対象ボード: MPG-2314

HPT入力ポートにUIN0を指定します。

関連: HOME

see also XIN0

UIN1

書式

UIN1

使い方

HPT(UIN1)

機能

HPT入力指定

解説

対象ボード: MPG-2314

HPT入力ポートにUIN1を指定します。

関連: HOME

see also XIN1

UP_DWN

書式

UP_DWN

使い方

INSET UP_DWN

機能

カウンタ入力設定

解説

対象ボード: MPG-2314

入力カウンタをアップダウンカウンタにする

INSET UP_DWN /* Set the counter to 'UP/DOWN' mode

U_A

書式

U_A

使い方

RMVS U_A 1000

機能

U軸指定

解説

対象ボード: MPG-2314/MPC-1200

RMVS等のPGコマンドでの軸指定コマンドです。

see also X_A

U_C

書式

U_C

使い方

stps U_C 1000

機能

カウンタ指定

解説

対象ボード: MPG-2314

U軸カウンタを指定します。

see also X_C

U_E

書式

U_E

使い方

RR(U_E)

機能

U軸エラー指定

解説

RR()関数の引数として使用し、移動後のU軸のエラーの有無を調べます。
0でなければ、なんらかのエラー要因が発生していることを示しています。
エラーの詳細は、LMT関数,PGE関数で調査します。

対象ボード: MPG-2314

see also X_E

VOID

書式

VOID

使い方

MOVL VOID 1000 2000 VOID

機能

入力無効

設定解除

解説

対象ボード: MPG-2314/MPC-1200他

パルス発生コマンド、I/Oコマンドの設定解除やSELECT_CASE 等に使われます。

```
INSET ALL_A VOID /* INSET conditions clear
MOVL 1000 0 0 VOID /* Z axis disable
STOP ALL_A VOID /* STOP conditions clear
INTA_ON VOID /* INTA_ON disable
INTA_OFF VOID /* INTA_OFF disable
INTB_ON VOID /* INTB_ON disable
INTB_OFF VOID /* INTB_OFF disable
----
SELECT_CASE VOID /* SELECT_CASE condition
TMOUT VOID /* TMOUT disable
TIMER(VOID|3) /* TASK3 timer_=0
PULSE_OUT VOID /* PULSE_OUT disable
SENSE_ON VOID /* SENSE_ON disable
SENSE_OFF VOID /* SENSE_OFF disable
CU_POST VOID /* CU_POST monitor
```


VOID_U

書式

VOID_U

使い方

movl P(1) VOID_U

機能

無効軸指定

解説

対象ボード: MPG-2314

U軸を除く軸を指定

X_A|Y_A|Z_A と同じ

MOVL P(1) VOID_X /* X axis doesn't move
MOVL P(2) VOID_Z /* Z axis doesn't move
JUMP P(3) VOID_Z /* It stops right above the P(3)

VOID_X

書式

VOID_X

使い方

```
movl P(1) VOID_X
```

機能

無効軸指定

解説

対象ボード: MPG-2314

X軸を除く軸を指定

Y_A|U_A|Z_A と同じ

see also VOID_U

VOID_Y

書式

VOID_Y

使い方

```
movl P(1) VOID_Y
```

機能

無効軸指定

解説

対象ボード: MPG-2314

Y軸を除く軸を指定

X_A|U_A|Z_A と同じ

see also VOID_U

VOID_Z

書式

VOID_Z

使い方

```
movl P(1) VOID_Z
```

機能

無効軸指定

解説

対象ボード: MPG-2314

Z軸を除く軸を指定

X_A|Y_A|U_A と同じ

see also VOID_U

VRING

書式

VRING

使い方

RANGE VRING|X_A 999

機能

リングカウンタ設定

解説

MPG-2314の現在位置カウンタをRINGカウンタに指定します。

たとえば、サンプルのように指定した場合、0～999までは増加し、999を超えると0に戻ります。

ターレット機構などで、有用な機能です。

RANGE VRING|X_A 999

XIN0

書式

XIN0

使い方

HPT(XIN0)

機能

HPT入力指定

解説

対象ボード: MPG-2314

HPT入力ポートにXIN0を指定します。

関連: HOME

```
100 IF HPT(XIN0) != 0 THEN /* If IN0(near-org) is on
110 RMVS X_A 10000 /* Moving to opposite direction to HOME
120 END_IF
130 WAIT RR(X_A) == 0
```

XIN1

書式

XIN1

使い方

HPT(XIN1)

機能

HPT入力指定

解説

対象ボード: MPG-2314

HPT入力ポートにXIN1を指定します。

関連: HOME

```
*HOME
IF HPT(XIN0)==1 THEN : RMVS X_A 5000 : END_IF
IF HPT(YIN0)==1 THEN : RMVS Y_A 5000 : END_IF
IF HPT(ZIN0)==1 THEN : RMVS Z_A -5000 : END_IF
WAIT RR(ALL_A)==0
SHOM X_A|Z_A|Y_A IN0_ON
HOME -100000 -100000 0 100000
WAIT RR(ALL_A)==0
```

X_A

書式

X_A

使い方

RMVS X_A 1000

機能

X軸指定

解説

対象ボード: MPG-2314/MPC-1200

RMVS等のPGコマンドでの軸指定コマンドです。

```
ACCEL X_A 30000 1000 500 /* Acceleration/deceleration setting
FEED X_A 100 /* Speed setting
INSET X_A MD_2PLS|ALM_OFF|LMT_OFF /* In port set
SHOM X_A IN0_ON /* Setting Return to the Origin
MOVS X_A 1000 /* Absolute coordinate movement
RMVS X_A 1000 /* Relative coordinate movement
STOP X_A STP_D /* Moving stop with deceleration
WAIT RR(X_A)==0 /* Wait until moving complete
IF LMT(X_A,LMTp)|LMT(X_A,LMTn)!=0 THEN /* Confirming reason for stop
etc
```


X_C

書式

X_C

使い方

stps X_C 1000

機能

カウンタ指定

解説

対象ボード: MPG-2314

Xカウンタを指定します。

```
10 PG 0
20 STPS X_C 1234 /* set the X counter
30 PRINT X(-1) /* display the X-counter value
#RUN

1234
-----
a=COMP_C(16,X_C) /* compare the COMP+ register to X counter
```

X_E

書式

X_E

使い方

RR(X_E)

機能

X軸エラー指定

解説

RR()関数の引数として使用し、移動後のX軸のエラーの有無を調べます。
0でなければ、なんらかのエラー要因が発生していることを示しています。
エラーの詳細は、LMT関数,PGE関数で調査します。

対象ボード: MPG-2314

```
100 MOVX X_A 10000
110 WAIT RR(X_A)=0
120 IF RR(X_E)!=0 THEN /* Confirming error status
130 PRINT "ERROR STOP"
140 ELSE
150 PRINT "NORMAL STOP"
160 END_IF
170 PRX RR(X_E)
```

YIN0

書式

YIN0

使い方

HPT(YIN0)

機能

HPT入力指定

解説

対象ボード: MPG-2314

HPT入力ポートにYIN0を指定します。

関連: HOME

see also XIN0

YIN1

書式

YIN1

使い方

HPT(YIN1)

機能

HPT入力指定

解説

対象ボード: MPG-2314

HPT入力ポートにYIN1を指定します。

関連: HOME

see also XIN1

Y_A

書式

Y_A

使い方

RMVS Y_A 1000

機能

Y軸指定

解説

対象ボード: MPG-2314/MPC-1200

RMVS等のPGコマンドでの軸指定コマンドです。

see also X_A

Y_C

書式

Y_C

使い方

stps Y_C 1000

機能

カウンタ指定

解説

対象ボード: MPG-2314
Yカウンタを指定します。

see also X_C

Y_E

書式

Y_E

使い方

RR(Y_E)

機能

Y軸エラー指定

解説

RR()関数の引数として使用し、移動後のY軸のエラーの有無を調べます。
0でなければ、なんらかのエラー要因が発生していることを示しています。
エラーの詳細は、LMT関数,PGE関数で調査します。

対象ボード: MPG-2314

see also X_E

ZIN0

書式

ZIN0

使い方

HPT(ZIN0)

機能

HPT入力指定

解説

対象ボード: MPG-2314

HPT入力ポートにZIN0を指定します。

関連: HOME

see also XIN0

ZIN1

書式

ZIN1

使い方

HPT(ZIN1)

機能

HPT入力指定

解説

対象ボード: MPG-2314

HPT入力ポートにZIN1を指定します。

関連: HOME

see also XIN1

Z_A

書式

Z_A

使い方

RMVS Z_A 1000

機能

Z軸指定

解説

対象ボード: MPG-2314/MPC-1200

RMVS等のPGコマンドでの軸指定コマンドです。

see also X_A

Z_C

書式

Z_C

使い方

stps Z_C 1000

機能

カウンタ指定

解説

対象ボード: MPG-2314
Zカウンタを指定します。

see also X_C

Z_E

書式

Z_E

使い方

RR(Z_E)

機能

Z軸エラー指定

解説

RR()関数の引数として使用し、移動後のZ軸のエラーの有無を調べます。
0でなければ、なんらかのエラー要因が発生していることを示しています。
エラーの詳細は、LMT関数,PGE関数で調査します。

対象ボード: MPG-2314

see also X_E

時間管理

Category

DS_SEC

書式

DS_SEC n

使い方

DS_SEC 5

機能

1 秒カウンタ停止

解説

nで指定した 1 秒カウンタSEC(n)を、停止します。

EN_SEC

書式

EN_SEC n

使い方

EN_SEC 1

機能

1 秒カウンタのカウント・イネーブル

解説

nで指定した 1 秒カウンタSEC(n)、をカウント・モードにします。

LIFE_TIME

書式

LIFE_TIME [val]

使い方

LIFE_TIME 100

機能

タイムスライス時間制御

解説

MPC-2000のタイムスライスはデフォルトで3msecですが、アプリケーションによっては、この時間を調整したほうがよい場合があります。

LIFE_TIME コマンドをこの時間を10 μ 秒単位で、500 μ 秒から5msecの間で設定することができます。

LIFE_TIME 250 --> 2.5msecの意味です。

パワーオンリセットでデフォルトに戻りますので、変更が必要な場合はプログラムに記述します。

また、引数無しの場合、現在のタイムスライス時間を返します。

SEC

書式

SEC MBK(n)
SEC n h m s

使い方

SEC MBK(7000)
SEC 7 17 20 2
SEC 8 0

機能

1 秒カウンタの初期設定

解説

1 秒カウンタSEC(0) ~ SEC(15)の設定を行います。

SEC 8 0

1 秒カウンタSEC(8)をクリアします。

SEC 7 17 20 2

1 秒カウンタSEC(7)を17時間20分2秒にセットします。

SEC MBK(7000)

カウンタ値のMBKへの複写位置を決定し、複写をイネーブルします。

```
SEC MBK(7000)
SEC 5 10 58 40
SEC 6 16 10 1
SEC 7 17 20 2
SEC 8 0
FOR i=5 TO 8
  EN_SEC i
```

```
NEXT i
FORK 11 *mon
END
*mon
DO
TIME 1000
FOR i=5 TO 8
SEC i
PRINT MBK(7000+(i*3)) MBK(7001+(i*3)) MBK(7002+(i*3))
NEXT i
PRINT "next"
LOOP
```

SET_RTC

書式

```
set_rtc arg  
set_rtc arg1 arg2 [arg3]
```

使い方

```
SET_RTC &H20000119  
SET_RTC &H00113000  
SET_RTC 2007 12 19  
SET_RTC 10 29 40
```

機能

RTCの時間を設定する。

解説

日付と時間を設定します。10進形式では引数を3個入力します。

```
SET_RTC 2007 12 19
```

```
SET_RTC 10 29 40
```

ヘキサ形式では引数は一つで以下のフォーマットです。

```
#set_rtc &h20070731
```

2007年07月31日に設定

```
#set_rtc &h182200
```

18時22分00秒に設定

設定された時間は、DATE(0),TIME(0)で参照します。

なお、SET_RTCコマンドは、FREEZEコマンドを実行して、保護・秘密化された状態では、実行できなくなります。実行できるはFREEZEで保護されたプログラム中のSET_RTCとなります。また、カレンダーICは、バッテリー・ダウンを検出すると、2130/01/01にプリセットされます。

```
SET_RTC &H20000119
SET_RTC &H00113000
PRX DATE(0) TIME(0)
SET_RTC 2007 12 19
SET_RTC 10 29 40
PRX DATE(0) TIME(0)
S_MBK DATE(0) 1000
S_MBK TIME(0) 1003
PRINT MBK(1002) MBK(1001) MBK(1000)
PRINT MBK(1005) MBK(1004) MBK(1003)
```

TIME

書式

TIME arg

使い方

TIME 100

機能

指定msecタスクを停止します。

解説

TIMEはタイミングをとるコマンドであると同時に実行効率を向上させるコマンドです。

TIMEで時間待ちをしている時間は、そのタスクはSLEEPとなり、CPUの時間資源を他のタスクに割り当てることができます。

TMOUT

書式

TMOUT n [taskn]

使い方

TMOUT 100
TMOUT 100 n
TMOUT VOID

機能

タイムアウト時間設定

解説

タイムアウト時間はmsec単位で設定します。設定できる最小時間は10msecです。

(対象:WS0(),WS1(),HOME)

デフォルトでは、13日(20000秒)が設定されています。

2番目の引数はタスク指定です。省略すると自己タスクに対する指定となります。

引数にVOIDを指定すると、初期値の20000秒がセットされます。

なお、TMOUTで設定された値は、WS0,WS1,HOMEの中でtimer__nに設定されます。

このため、WS0,WS1,HOMEの直前で、timer__nの値を操作しても無効です。

引数なしのTMOUTコマンドで現状の設定値を表示させることができます。

```
#TMOUT
TMOUTs
0 10000
1 10000
2 2000000
3 2000000
4 2000000
5 2000000
6 2000000
```

7 2000000
8 2000000
9 2000000
10 2000000
11 2000000
12 2000000
13 2000000
14 2000000
15 2000000
#

DATE

書式

DATE(0)
DATE(255)
DATE(VOID)

使い方

```
IF DATE(0)==&H20070731 THEN  
  PRINT "HAPPY BIRTHDAY"  
END_IF
```

機能

年月日取得

解説

ヘキサ形式で日付値を得ます。
引数をいれると、引数と論理積をとって値を返します。
引数にVOIDを設定すると、10進で値を返します。
なお、年月日設定は SET_RTC コマンドで実施します。

```
IF DATE(0)==&H20070731 THEN  
  GOTO *Thisday  
END_IF  
PRX DATE(0)
```


SEC

書式

SEC(n)

使い方

```
IF SEC(0)>SEC(1) THEN : print "TIME_OVER" : END_IF
```

機能

1 秒カウンタ

解説

1 秒カウンタは、SEC(0)～SEC(15)の15個用意されています。

SEC(n)は、パワオンリセット後、カウントは停止しています。

EN_SEC nによってカウントが再開されます。

カウンタの初期化は、SECコマンドで行います。

SEC n 0でクリア。SEC n 10 9 8 で10時間9分8秒です。

SEC(n)のデータは、時間(2byte) 分(byte) 秒(byte)の4byte形式となっており、直接値を見ることはできません。

参照するには以下の演算が必要です。

```
print SEC(0)/65536 -->時間
```

```
print SEC(0)/256&255 -->分
```

```
print SEC(0)&255 --> 秒
```

時間アラームとして用いるには、

```
SEC 10 11 12 15
```

```
IF SEC(9)>SEC(10) THEN
```

というように、使用していないカウンタに値をいれて比較する方法が見通しがよくなります。

サンプルは、SEC(1)固定値30秒とし、SEC(0)をインクリメントさせて比較し、指定時間を超えたらWAITから抜け出るといったものです。

```
10 SEC 1 0 0 30
20 SEC 0 0 0 0
30 EN_SEC 0
40 DS_SEC 1
50 WAIT SEC(0)>SEC(1)
60 PRINT "time_out" SEC(0)
#run

time_out 31
#
```

TIME

書式

TIME(0)
TIME(255)
TIME(VOID)

使い方

IF TIME(0)

TIMEOUT

書式

TIMEOUT(n)

使い方

WAIT SW(1)&SW(2) OR TIMEOUT(0)

機能

timer_のタイムアウト判別

解説

n==0の時、(timer_==0)と同じ意味。

WAIT SW(1)&SW(2) OR TIMEOUT(0)という記述により、意味が明確になる。

なお、TIMEOUT()関数は他のタスクのtimer_も評価できる。

n : -1 タスク0のtimer_の0判別

n : 1 ~ 31 タスクnのtimer_の0判別

```
timer_=10
PRINT TIMEOUT(0)
WAIT SW(1)&SW(2) OR TIMEOUT(0)
PRINT TIMEOUT(0)
```

TIMER

書式

TIMER(arg)

使い方

```
a=TIMER(3)
a=TIMER(VOID|3)
a=TIMER(1,1)
```

機能

timer_ を参照

解説

timer_はタスク変数のため、他のタスクから値を参照したり設定することができません。

TIMER(n)は引数にタスク番号を指定すると、そのタスクのtimer_の値を得ることができます。注)単位は0.1秒です。

また、タスク番号にVOIDを論理和すると、そのタスクのtimer_を0にできます。

TMOUTコマンドで管理されるタイムアウトは、このtimer_変数を使っていますので、他のタスクから強制タイムアウトさせるには、この関数を使って、timer_を0にします。

TIMER(1,n)という形式で引数を与えると、タスクnのTMOUT設定値を呼び出す事ができます。

SEC

書式

SEC

使い方

pr SEC

機能

1 秒カウンタ

解説

1 秒ごとにアップカウントする変数です。

```
10 SEC=0
20 PRX TIME(0)
30 WAIT SEC>10
40 PRX TIME(0)
#RUN
```

```
00022538
00022548
```

SYSCLK

書式

SYSCLK

使い方

pr SYSCLK

機能

システムクロック

解説

パワーオン後約1msecごとにインクリメントする変数です(CPUのクロック基準)

```
10 SYSCLK=0 /* SYSCLK clear
20 TIME 100 /* delay 100msec
30 a=SYSCLK
40 PRINT a /* display
RUN
```

101

timer_

書式

timer_

使い方

IF timer_==0 THEN

機能

ダウンカウンタ

解説

タスク変数

0.1秒ごとにダウンカウントして0で停止します。

```
10 timer_=100 /* set 10Sec -> 0.1Sec count down
20 PRX TIME(0) /* display current time
30 WAIT timer_==0 /* wait 10sec
40 PRX TIME(0) /* display current time
#run

00014841
00014851

-----

10 FORK 3 *JOB
20 SYSCLK=0 /* SYSCLK init
30 TIME 100
40 WAIT TIMER(3)==0 /* wait "timer_" of the TASK3 == 0
50 PRINT SYSCLK "mSec"
60 END
70 *JOB /* TASK3
80 timer_=100 /* set 10Sec -> 0.1Sec count down
90 DO : SWAP : LOOP
```


#RUN

9995 mSec

制御文

Category

ON_ERROR

書式

ON_ERROR arg

使い方

ON_ERROR *USB
ON_ERROR VOID

機能

エラー処理ジャンプ先を定義

解説

コマンド、関数などでエラーが発生した場合、通常、実行中のプログラムが停止します。

ON_ERROR

コマンドは、プログラムを停止させず、エラー処理プログラムを指定して、プログラムの実行を継続させます。方法は以下のとおりです。

ON_ERROR *labelで飛び先を規定します。

規定を解除する場合は、ON_ERROR VOIDを実行します。

ON_ERRORはどのようなエラーでもエラー処理に制御を移してしまうため、

エラー処理プログラムでは、適切にエラーコードによって処理を分類する必要があります。

通常、実行時に発生するエラーは、致命的なものがほとんどで、リトライ措置は不可能です。

この場合は、エラー場所と内容を外部に知らせてデバッグに役立てます。

しかし、USBメモリをアクセスするような場合は、接続デバイスの状態によりランタイム・エラーが発生することがあります。

この場合は、RST_USBなど適切な処理により、デバイスを正常化してプログラムを再継続することができます。

ON_ERRORによるエラー処理ルーチンから、通常処理プログラムへの戻りにはGOTO,RESUMEを使用します。

GOTOの場合は、エラー発生場所がサブルーチン中の場合、戻し場所に注意してください。

RESUMEの場合は発生箇所に戻すため、コマンドをリトライする場合は、RESUME,

リトライさせずに次の処理に移るには、RESUME_NEXTと記述します。

エラーコードは、タスク変数、err_に反映されます。err_の値に適合した処理を記述します。
err_は上位1byteがエラーコード、下位3byteがプログラム番号となっています。

err_>>24 --> エラーコード

ERR\$(err_) --> エラーメッセージ

err_&&HFFFFFF --> プログラム番号

エラー番号はエラーメッセージの末尾に表示されます。
以下はUSBメモリ関連のエラーコードです。

このUSBは使用中です。:53

USBメモリがありません。:54

MRS-MCOMがありません。:55

USBメモリが動作異常。:56

```
FORK 1 *case1
  TIME 500
  FORK 2 *case2
  END
*case1
  ON_ERROR *err1
  DO
    S_MBK 1 9000
    PRINT 10
    PRINT 20
  LOOP
*err1
  PRINT "case1=" TASKn err_&&H00FFFFFF ERR$(err_) err_>>24
  TIME 1000
  RESUME _NEXT
  END
*case2
  ON_ERROR *err2
  DO
    OUT 1 -10000
    PRINT 1
    PRINT 2
  LOOP
*err2
  PRINT "case2=" TASKn err_&&H00FFFFFF ERR$(err_) err_>>24
  TIME 1000
  RESUME
  END
```

RESUME

書式

RESUME [arg]

使い方

RESUME
RESUME _NEXT

機能

エラー処理から戻る

解説

ON_ERRORからの戻り処理です。
RESUMEは発生箇所に戻すため、コマンドをリトライする場合は、RESUME,
リトライさせずに次の処理に移るには、RESUME _NEXTと記述します。
ON_ERRORを参照してください。

BREAK

書式

BREAK

使い方

```
DO  
IF SW(0)==1 THEN : BREAK : END_IF  
LOOP
```

機能

FOR-NEXT,DO-LOOP,WHILE-WENDの繰り返し実行のキャンセル

解説

複数の条件でエンドレス実行をキャンセルさせる場合は、DO-LOOPで記述しておき、IF文でBREAKを実行するほうが、明快な表現となる。

BREAK文は、ループ中でどこにでも複数記述できる。

CANCEL_RETURN

書式

CANCEL_RETURN

使い方

CANCEL_RETURN : GOTO *AAAA

機能

RETURNスタックの破棄

解説

禁止手です。

RETURNスタックを破棄します。

サブルーチンからRETURN文でもどらず、親ルーチンのラベルなどに直接飛ばす場合などに使用します。むやみに使うべきではありません。

```
FOR i=1 TO 100
  s=0
  GOSUB *aho
NEXT
PRINT "normal" i j s
END
*baka
PRINT i j s
GOTO *init
*aho
FOR j=0 TO 100
  s=s+j
  IF j=50 THEN : CANCEL_RETURN : GOTO *baka : END_IF
NEXT j
RETURN
```

DO-LOOP

書式

```
DO  
LOOP
```

使い方

```
DO  
ON 0 : TIME 1 : OFF 0  
LOOP
```

機能

エンドレス繰り返し実行

解説

DOからLOOPの間をエンドレス繰り返し。繰り返しを停止させる場合は、BREAK文を用いる。

```
DO  
IF SW(0)==1 THEN : BREAK : END_IF  
LOOP
```


END

書式

END

使い方

機能

実行終了

解説

プログラム終了。マルチタスク・プログラムの終了。

タスク0の場合は入力待ちプロンプトを出力。マルチタスク・プログラムの場合は、タスク停止となる。

FOR-NEXT

書式

```
FOR var=arg1 TO arg2 [STEP arg3]
```

使い方

```
FOR i=0 TO 15 STEP 2  
ON i : TIME 100 :OFF i  
NEXT
```

機能

インクリメントあるいはデクリメント繰返処理

解説

決まった回数の繰返し処理に使用する制御文です。NEXTには変数名をいれる必要はありませんが、変数名がはっていると、FOR文で指定された変数名と整合を確認します。

GOSUB,GOSUB_NE

書式

```
GOSUB *Label [arg1,arg2..]
```

使い方

```
GOSUB *Label  
GOSUB *Label arg1 arg2 ..
```

機能

サブルーチン・コール

解説

サブルーチンは、スタック・メモリを用いているので、GOSUBで呼ばれたプログラムは必ずRETURNでもどる必要があります。

サブルーチン中からRETURNを用いずGOTOなどで呼び出し元に戻すプログラムを作ってしまうと、スタック・オーバフローやアンダーフローというエラーになりプログラム実行が停止します。

BL/1のGOSUBコマンドは、呼び出し先ラベルの後に、サブルーチンに引き渡す引数を追加することができます。この引数値は、サブルーチン側で、_VARコマンドで取り出します。

_VARの引数に、タスクローカル変数を用いるとサブルーチンは汎用的になります。

GOSUBの代わりにGOSUB_NEを用いると、呼び出し先ラベルが存在していなくてもコンパイル・エラーとならず、そのまま実行可能です。

飛び先ラベルの無いGOSUB_NEは、実行時には無視されます。

```
10 GOSUB *CAL 300 400  
20 _VAR RES  
30 PR RES  
40 END  
50 *CAL
```

```
60 _VAR V_ W_  
70 RETURN SQR(SQ(W_)+SQ(V_))  
RUN
```

*

Compiling

500

#

GOTO

書式

GOTO *Label

使い方

```
IF A==1 THEN : GOTO *ERR : END_IF  
GOTO *LOOP
```

機能

無条件分岐

解説

指定されたラベルに実行制御を移します。

IF-THEN-ELSE-END_IF

書式

IF arg THEN

使い方

```
IF SW(0)==1 THEN : ON 0 : END_IF  
IF SW(0)==1 THEN : ON 0 : ELSE : ON 1 : END_IF
```

機能

条件分岐

解説

argが0でない場合THENの後ろを実行。0の場合は、ELSEの後ろを実行するか、END_IFの後ろへジャンプ

RETURN

書式

RETURN [arg1,arg2..]

使い方

GOSUB *LABEL

*LABEL

RETRUN

*LABEL

RETURN aho

機能

サブルーチンから戻る。また、引数をGOSUBを実行した側に戻ることができる。

解説

サブルーチンからGOSUB呼び出しをしたプログラムにもどる。

GOSUBで呼ばれたプログラムは必ずRETURNで戻らなければならない。

また、RETURNに引数を与えると、結果を親プログラムに戻ることができる。

```
10 GOSUB *CAL 300 400
20 _VAR RES
30 PR RES
40 END
50 *CAL
60 _VAR V_ W_
70 RETURN SQR(SQ(W_)+SQ(V_))
RUN
```

*

Compiling

500

#

RUN

書式

RUN arg1

使い方

RUN
RUN *LABEL
RUN 900

機能

プログラム実行

解説

デバッグ時のプログラム実行です。

プログラムをコンパイルの後、フラッシュROMに保存して実行します。

プログラムがすでにコンパイル済みの場合はただちに実行します。

初期状態のMPCに既存のプログラムを読み込んで稼動する際にも、読込後RUNを実行してください。

プログラミングケーブル未接続状態で電源を入れると自動実行になります。

SELECT_CASE

書式

SELECT_CASE arg

使い方

```
SELECT_CASE IN(0)&&HF  
CASE 1 : GOSUB *A  
CASE 2 : GOSUB *B  
CASE_ELSE GOSUB *C  
END_SELECT
```

```
SELECT_CASE VOID  
CASE SW(1) : GOSUB *A  
CASE SW(2) : GOSUB *B  
CASE_ELSE GOSUB *C  
END_SELECT
```

機能

CASE 文中の数値による分類分岐

CASE 文中の論理式による分類分岐

解説

SELECT_CASE は、与えられた引数とCASE
の引数を比較して、一致したCASE文の後ろのみを実行する排他的分類制御。【EXAM 1】

ただし、SELECT_CASEの引数をVOIDとすると、CASE文独自の論理式を評価して実行するようになる。CASE文の評価は、上から順に行われる。

また、CASE文中の論理式はAND,OR等論理接続詞をもちいることができる。【EXAM 2】

CASE 1 : CASE 2 というようにCASE2文を連続して並べるとそれぞれのCASE文の論理ORとなる。【EXAM 3】

【EXAM 1】

```
SELECT_CASE a
CASE 1 : PRINT 1
PRINT 111
CASE 2 : PRINT 3
PRINT 123
CASE_ELSE : PRINT 4
PRINT 456
END_SELECT
```

【EXAM 2】

```
SELECT_CASE VOID
CASE SW(192)==1 : PRINT 192 : WAIT SW(192)==0
CASE SW(193)==1 : PRINT 193 : WAIT SW(193)==0
CASE SW(194)==1 : PRINT 194 : WAIT SW(194)==0
CASE_ELSE
END_SELECT
```

【EXAM 3】

```
SELECT_CASE A
CASE 0
CASE 1 : PRINT 1
CASE 2 : PRINT 2
CASE 5 : PRINT 5
CASE_ELSE : PRINT 3
END_SELECT
```

```
SELECT_CASE VOID
CASE A==0
CASE A==1 : PRINT 1
CASE A==2 : PRINT 2
CASE A==5 : PRINT 5
CASE_ELSE : PRINT 3
END_SELECT
```

WAIT

書式

WAIT logical_eqations

使い方

WAIT SW(0)==1
WAIT SW(-2)

機能

条件待ち

解説

条件式が1になるのを待ちます。

WAIT SW(0)==0 SW(0)が0になるのを待つ

WAIT SW(-2) SW(-2)が1になるのを待つ

WAIT A==100 A==100の論理式が1になるのを待つ。つまり、Aが100になるのを待つ

また、条件式にはAND,OR等の接続詞を使用できます。タイムアウトが必要な場合には、

WAIT SW(1)&SW(2) OR TIMEOUT(0)

のように、OR TIMEOUT(0)追加するとタイムアウト処理も可能になります。

WAITに"UNTIL"を追加すると、リアルタイム対応となります

例えば、WAIT UNTIL HSW(192)==1

という記述では、このコマンドを実行したタスクは休止状態となり、かわってOSが条件式を実行します。

タスク切り替えのタイミングに、条件式が成立していれば、休止タスクを実行状態にもどし、そのタスクに実行権を渡します。

これにより、反応速度は、起動タスク数に依存せず、タイムスライスの時間（3msec）以内となります。

ただし、UNTIL指定のWAIT文はOSへの負担が大きく、数多くのタスクがWAIT

UNTILで条件待ちをすると、かえって処理が遅くなる場合もあります。

WAIT UNTILは緊急性の高い条件検出に対して使用してください。

この応答時間をより高速にするには、LIFE_TIMEコマンドでタイムスライスの時間を短くします。

```
10 timer_=10
20 WAIT SW(1)&SW(2) OR TIMEOUT(0)
30 IF TIMEOUT(0) THEN : GOTO *TIME_OUT : END_IF
```

WHILE-WEND

書式

WHILE 論理式 ~ WEND

使い方

```
WHILE SW(0)==1  
ON 0 : TIME 1 : OFF 0  
WEND
```

機能

条件付無限ループ実行

解説

条件を定めて繰り返し実行する場合に使用します。

論理式の値が1である限りWHILEとWENDの間のプログラムを実行しつづけます。

演算

Category

CONST

書式

CONST var val

使い方

CONST A_P 123

機能

変数の定数化

解説

変数を定数化して変更できないようにします。

DIM

書式

DIM label(val)
DIM label(val1,val2)
DIM label1(val) label2(val) label3(val) ...

使い方

DIM A(100)
DIM array(100,100)
DIM A(100) B(100) C(5)

機能

配列要素の宣言

解説

配列は 1 次元、2 次元のいずれかで、合計20000データの範囲で自由に宣言できます。
ラベルは 1 5 文字以内で64個までです。一度、配列が64個を超えるとその後のラベル管理ができなくなりますので、プログラム修正後、再ロードしてください。

DIMCPY

書式

DIMCPY arg1 arg2 count

使い方

```
DIMCPY 1000 U(3) 60
DIMCPY X(1) aho(10) 10
DIMCPY MBK(1) Y(4) 50
DIMCPY X(3) MBK(200) 60
DIMCPY X(3) MBK(200-Lng) 60
```

機能

MBK() X(),Y(),Z(),U() および定義した配列要素間でデータの転写。

解説

MBK() X(),Y(),Z(),U() およびDIM定義した配列要素間でデータの転写を実施。

DIMCPY MBK(1) MBK(100) 50のように同一の要素でもエリアが重複していなければ転写可能です。

なお、DIMCPYでMBKデータは、ワード型としてのみ扱います。

MBKデータをロング型とする場合は、-Lngを付加します。

ただし、DIMCPY MBK(1) MBK(100-Lng) 50のような場合は、元も先もロング型扱いとします。

```
DIM aho(300)
DIM baka(300)
DIMCPY 1010 aho(3) 25
FOR i=1 TO 30
  PRINT i aho(i)
NEXT
S_MBK 100 50 30
DIMCPY 12345 MBK(52) 10
PR "MBK"
FOR i=50 TO 65
  PRINT i MBK(i)
NEXT
```

```
*test2
FOR i=1 TO 50
  aho(i)=i*-1000
NEXT i
DIMCPY aho(1) baka(100) 30
PR "BAKA()"
FOR i=90 TO 135
  PRINT i baka(i)
NEXT
NEWP
DIMCPY baka(100) X(10) 20
DIMCPY baka(100) Y(11) 20
DIMCPY baka(100) Z(12) 20
DIMCPY baka(100) U(13) 20
PR "X()"
FOR i=5 TO 30
  PRINT i P(i)
NEXT
DIMCPY X(10) MBK(52) 20
PR "X->MBK"
FOR i=50 TO 65
  PRINT i MBK(i)
NEXT
DIMCPY MBK(52) baka(70) 20
PR "X->MBK"
FOR i=65 TO 95
  PRINT i baka(i)
NEXT
```

FILL

書式

FILL array(N) Count [Val Inc]

使い方

```
FILL aho(0) 0 0  
FILL aho(10) 10 -110 2  
FILL X(6) 20 10000 100  
FILL P(6) 20 100 0  
FILL MBK(100) 10 500 -2  
FILL MBK(200-Lng) 100000 10000
```

機能

配列要素にデータを連続設定する。点データ、MBKデータにも有効

解説

配列要素の初期化コマンドです。

第1引数は初期化したい配列の先頭要素を記述します。

第2引数は、カウント数です。いくつ、初期化するかを指定します。

ただし、0を指定すると、指定配列すべてとなります。

例えば、

```
FILL AHO(0) 0
```

とするとAHO(0) ~ 配列範囲内で内容をすべて0に設定します。

```
FILL AHO(5) 10
```

この場合は、AHO(5) ~ AHO(14)を0にするという意味になります。

第3引数をいれると、0以外の数を設定できます。

```
FILL AHO(5) 10 100
```

AHO(5) ~ AHO(14)を100に設定するという意味になります。

さらに、第4引数をいれると設定値をオートインクリメントします。

FILL SYSDAT(1) 100 501~Lng 2

この例では、SYSDAT(1) ~ SYSDAT(100)に

501~Lng

503~Lng

と2を加えながら、設定していきます。負の数を設定すれば、減算しながら設定します。

配列要素は、バッテリバックアップされていますが、プログラムの変更などで、メモリ位置が変わります。このため、常に適切な初期化が必要となります。

配列要素に点データP(n)を与えると、X,Y,Z,Uのすべての要素を同じ値にします。

FILL P(1) 100 => P(1) ~ P(100)を0,0,0,0にします。

```
DIM aho(100)
FILL aho(0) 0 0
FILL aho(10) 10 -110 2
FOR i=8 TO 30
PRINT i aho(i)
NEXT
FILL X(6) 20 10000 100
FILL MBK(100) 10 500 -2
FILL MBK(200~Lng) 100000 10000
```

INC

書式

INC var [Val]

使い方

INC A
INC A -10

機能

変数の増減(マルチタスク)

解説

マルチタスクで共有の変数を用いて増減を行うと、リードとセットがバラバラのタイミングになる場合があります、タスク間で変数の増減を正確に行えない場合があります。

INCコマンドはリード&セットをタスク内で完結させるため、そうした不具合がありません。

引数がない場合は単純な+1です。引数を追加すると、その値を変数に加えます。

SFTL

書式

SFTL ary(val)
SFTR MBK(n) TO MBK(m)

使い方

SFTL ary(5)
SFTL MBK(5) TO MBK(14)

機能

配列の左シフト

解説

SFTL ary(5)
ary(0) ~ ary(5) で左シフト
ary(5) -> ary(4) : ary(4) -> ary(3)

SFTL MBK(5) TO MBK(14)

MBK(5) ~ MBK(14)で左シフト
MBK(14) -> MBK(13) : MBK(13) -> MBK(12)

```
130 FOR i=0 TO 9
140 ary(i)=i*1000
150 NEXT i
160 FOR i=0 TO 9
170 PRINT i ary(i)
180 NEXT
190 PRINT "SHOW SFTL"
200 SFTL ary(5) --> 5->4 4->3 ~ 0 -> 5
210 FOR i=0 TO 9
```

220 PRINT i ary(i)
230 NEXT

SHOW SFTL

0 1000
1 2000
2 3000
3 4000
4 5000
5 0
6 6000
7 7000
8 8000
9 9000

SFTR

書式

SFTR array(val)
SFTR MBK(n) TO MBK(m)

使い方

SFTR array(5)
SFTR MBK(5) TO MBK(14)

機能

配列もしくは、MBKデータの右シフト

解説

SFTR array(5)
array(0) ~ array(5) で右シフト
array(1) -> array(2) : array(2) -> array(3)

SFTR MBK(5) TO MBK(14)

MBK(5) ~ MBK(14)で右シフト
MBK(5) -> MBK(6) : MBK(6) -> MBK(7)

```
10 DIM ary(5)
20 FOR i=0 TO 4
30 ary(i)=i*1000
40 NEXT
50 FOR i=0 TO 4
60 PRINT i ary(i)
70 NEXT
80 SFTR ary(3) : 'rotate ary(0)-ary(3)
90 PRINT "SHOW SFTR"
```

```
100 FOR i=0 TO 4
110 PRINT i ary(i)
120 NEXT
RUN
```

```
0 0
1 1000
2 2000
3 3000
4 4000
SHOW SFTR
0 3000
1 0
2 1000
3 2000
4 4000
```

_VAR

書式

`_VAR arg1 [arg2 ..]`

使い方

*TASK
`_VAR vala_ valb_`

機能

GOSUBもしくはRETURNで与えられた引数を取り出す。

解説

GOSUB文では引数を与えてサブルーチンを実行させることができます。

_VARはその引数を取り出して指定の変数に代入するコマンドです。

_VARはRETURN 文の引数も取り出すことができます。

@

書式

@(arg)

使い方

IF @(A==1) THEN
IF @((A!=1) & (B!=1)) THEN

機能

論理反転

解説

1->0 0->1に変換する論理反転です。
NOT()がロング型反転であるのに大して、@()は0か1しか返しません。

ABS

書式

ABS(arg)

使い方

A=ABS(-100)

機能

絶対値を得る

解説

引数を正の整数に変換して返す。

A=-123

A=ABS(A)

Aは123となります。

CK_Z,CK_NZ

書式

CK_Z(arg)
CK_NZ(arg)

使い方

```
IF SW(1)&SW(2)|CK_Z(A) THEN : PRINT "OK" : END_IF
```

機能

ゼロテスト、ノンゼロテスト

解説

CK_Z(arg) 引数の値が0であれば1,0でなければ、0を返します。
CK_NZ(arg) 引数の値が0であれば0,0でなければ、1を返します。

NOT

書式

NOT(arg)

使い方

A=NOT(1)

機能

引数のビット反転

解説

ロング型でのビットNOT

#prx NOT(&Hf)

FFFFFFF0

マルチタスク

Category

CONT

書式

CONT arg

使い方

CONT 8

CONT VOID|1

機能

SLEEPING 中のタスクを再開

解説

PAUSEコマンドによって一時停止しているタスクを再開します。

PAUSE (STP_D,n)によって一時停止させられた、WARP,JUMP,JMPZは、CONTによって、自動再実行されます。

ただ、状況によっては、不都合な場合もあります。この場合は、CONT VOID|n と実行します。

これにより一時停止された、WARP,JUMP,JMPZの実行はキャンセルされます。

FORK

書式

FORK n *LABEL

使い方

```
FORK 1 *LABEL  
END  
*LABEL  
DO  
LOOP
```

機能

タスク起動

解説

マルチタスクで、*LABELよりプログラムを実行します。
指定できるタスク番号は1～31です。
起動されたタスクはENDで終了するか、QUITで強制終了できます。
すでにFORKしているタスクをFORKすると、二重起動エラーとなります。
この場合、QUITしてから再起動するか、QUIT_FORKを用います。
なお、タッチパネル通信やCU_POSTコマンドはタスクを占有します。
これらのタスクに干渉しないようにコマンドを用いてください。

PAUSE

書式

PAUSE arg
PAUSE (STP_D,taskn)

使い方

PAUSE n

機能

タスクの一時停止

引数が、(STP_D,task)の場合 タスクの一時停止と停止コマンドの実行

解説

動作中のタスクをSLEEPING状態(無限タイマー停止)にします。CONTで再開します。

引数を (STP_D,task)のようになると、対象タスクを停止するとともに、STOP

STP_Dを実行し、なおかつ、JUMP,JMPZコマンドに対して、再実行フラグをたてます。

この場合、CONTコマンドにより再開されたタスクはWARP,JUMP,JMPZコマンド実行中であれば、再実行措置をとります。

ただしWARPのハンド操作機能は、PAUSEのかかった時点で有効無効が不明となるため、再開は要注意。

QUIT

書式

QUIT arg1 arg2 arg3..

使い方

```
QUIT 1  
FOR I=1 TO 4 : QUIT I : NEXT
```

機能

タスクの停止

解説

FORK によって起動されたマルチタスク・プログラムを停止する。

QUIT_FORK

書式

QUIT_FORK n *LABEL

使い方

QUIT_FORK 1 *LABEL

機能

タスク起動

解説

FORKと同じ機能ですが、相手タスクが既に起動されていてもエラーを発生しません。

SWAP

書式

SWAP

使い方

機能

実行中にプログラムを強制スワップ(実行タスク交替)

解説

長い処理時間を要するタスクを実行させていると、タスクのタイムスライス時間いっぱいまで時間を占有し他のタスクの実行が遅くなります。

こうした場合、人為的にSWAPを実行させることによって実行権を強制的に他のタスク移すことができます。

ON

書式

ON(n)

使い方

```
WAIT ON(-1)==0
PRINT "WATSHI HA " TASKn
OFF -1
'
IF ON(-1)==0 THEN
PRINT "WATSHI HA " TASKn
OFF -1
END_IF
```

機能

メモリIOのリード&セット(セマフォ)

解説

ON(n)はメモリI/Oおよび出力ポートに対して、ONコマンドと同様、ポートをONします。

関数値として、ONする直前の指定ポートの値を返します。

ON(n)でnをメモリI/Oに指定すると、ポートnがセマフォとなります。

nを通常出力ポート番号としても同様に使用できます。

```
OFF -1
FOR i=1 TO 10
  FORK i *test
NEXT
END
*test
WAIT ON(-1)==0
PRINT "WATSHI HA " TASKn
OFF -1
TIME SYSCLK%1000
```

GOTO *test

TASK

書式

TASK(n)
TASK(n|256)

使い方

WAIT TASK(1)!=0
a=TASK(1|256)

機能

タスクの状態を参照する。

解説

nはタスク番号です。

TASK(n)の戻り値

0: タスクは実行中。
1: タスクはタイマーにより待機している。
255: タスクは終了しているか、QUITされている。
引数に-1をいれると、自己タスク番号を返す。

TASK(n|256)の戻り値。

bit0: タスクはタイマーにより待機している。
bit1: タスクはPAUSEされている。

例)

0: タスクは実行中
1: TIME実行中
2: PAUSEされた
3: TIMEのときPAUSEされた
255: タスクは終了しているか、QUITされている。
(TASK 0はEND終了すると0が返されます)

TASKn

書式

TASKn

使い方

IF TASKn==0 THEN

機能

自己タスク番号取得

解説

実行中のタスク番号を得ます。TASKnはグローバル変数ですが、タスク・モニタにより、タスクに実行権がひきわたされるたびに、TASKnにタスク番号を書き込んでいます。

従って、TASKnをあやまって、他の値に変更しても、タスクが切り替わるごとに正常化します。

```
10 FORK 10 *SUBTASK
20 PRINT "main=" TASKn
30 END
40 *SUBTASK
50 TIME 500
60 PRINT "sub=" TASKn
70 END
#run
```

```
main= 0
# sub= 10
```

タッチパネル

Category

ADD_MBK

書式

ADD_MBK add_value adrs

使い方

add_mbk 1000 1

機能

MBK()配列の直接加算

解説

配列MBK()のデータを直接加算します。

```
#pr mbk(1)
1000
#add_mbk 1000 1
#pr mbk(1)
2000
#
```

MEWNET

書式

MEWNET arg1 [COMn] [mode]
MEWNET [COMn]

使い方

MEWNET 9600
MEWNET 9600
MEWNET 38400
MEWNET 38400 5
MEWNET 9600 1 B7O
MEWNET 38400 2 RS485
MEWNET 38400 1 COM

機能

タッチパネル用MEWNETプロトコル設定

解説

タスクをMEWNET(Panasonic電工FPシリーズコンピュータリンク)通信にわりあてて、MBK()配列とタッチパネルをデータ共有させます。

(WD,WC,RD,RCプロトコルによる共有)

どのタスクが割り当てられるかは、以下の規則に従って通信チャンネル番号で決定されます。

割り当てタスク = 32 - ch番号

このため、最初のユーザチャンネルCH1をMEWNETとして使用すると、タスク31を通信タスクとして占有します。

MRS-MCOMの最初のCH番号は3です。この場合は、 $32 - 3 = 29$ が、通信タスクに割り当てられます。

ボーレートは9600,19200,38400のうちから選択できます。

第2引数はRSのチャンネル番号で1~5を指定できます。(MPC-1000,N816は1,2のみ)

第3引数は、通信フォーマット設定です。通常は省略し、8ビットパリティ無し通信としますが、パリティやビット

ト数の必要な場合に追加します。

B7O: 7bit奇数パリティ

B7E: 7bit偶数パリティ

B8O: 8bit奇数パリティ

B8E: 8bit偶数パリティ

なお、MEWNETコマンドは、通信プロトコルの初期化を含みますので、CNFG#コマンドとは併用しないでください。

例

MPC-2200 CH2で接続 占有タスクは30

MEWNET 38400 2

MRS-MCOM #1 CH5で接続 占有タスクは27 (RS-232,RS-422とも、RS-485は未対応です。)

MEWNET 38400 5

MPC-2000 CH1で接続 占有タスクは31 Bit7 奇数パリティ (三菱小型タッチパネル)

MEWNET 9600 1 B7O

B7Eは Bit7 偶数パリティの場合

なお、MEWNETコマンドは一度実行されると、以後自動起動になります。このため、CHを変更する場合は、MEWNET [COMn]を実行して登録を抹消してください。

*MEWNET 1 の場合、COM1でのMEWNET停止。

MPCINITではすべての登録を初期化します。

--- ?????? GP2400 ??? ---

???? ? I/O??? ? ??????

???? 38400 (MEWNET?????????????)

???? 8

???????? 1

???????? ?

???? X??

???? RS232C

???? ? I/O??? ? ??????????

????????????(1-127) [10]? ? [1]? ??????????

引数の末尾にRS485を追加すると、RS485ポートでのMEWNET接続が可能になります。

しかし、MEWNETにはRS485が規定されていないため、MEWNETプロトコルで、RS485通信に対応しないタッチパネルもありますので注意してください。

なお、データの対応は以下のようになります。

DT0 ~ : MBK(0) ~

R0 ~ : ON/OFF/SW/IN/OUT 7YYXX ~ (DT7900 ~ DT7999と重複します)

YY=バンク番号(0 ~ 99) XX=ビット番号(0 ~ 15)

DTエリアは通常の番号対応ですが、Rエリアは70000以上の値で、下2桁がビット番号、中2桁がバンク番号となります。

IN/OUTの場合は、XXのビット番号を0とします。

MEWNETタスクは"\$"文字で始まる、CRでターミネートされたコマンドを即時実行する機能があります。

これを有効にするには、引数中に予約変数"COM"を加えます。

例 : MEWNET 38400 1 COM

S_MBK

書式

S_MBK arg1 arg2
S_MBK str adr c
S_MBK arg adr count

使い方

S_MBK 1 10
S_MBK 2 11
S_MBK 1000000 20-Lng
S_MBK a\$ 100 10
S_MBK 100 50 20
S_MBK 100-Lng
S_MBK DATE(0) 100

機能

タッチパネルデータ設定

解説

S_MBKはコマンドは、MBK配列に値を設定します。

1) 10番に1を設定 -> S_MBK 1 10

2) 20番に1000000を設定。 -> S_MBK 1000000 20-Lng
この場合MBK(20)に下位ワード、MBK(21)に上位ワードが入る。

3) 文字列の代入-> S_MBK a\$ 100 10 文字列("abc"も可)を100番地から10文字設定
文字列は文字列変数か文字列定数

4) 一括設定 -> S_MBK arg adr count
MBK(adr) ~ MBK(adr + count -1)に値argを一括代入します。

5) 表示 --> S_MBK nとするとMBK(n) ~ 内容表示。Lng表示は、S_MBK n-Lngとする。

S_MBK DATE(0) n / S_MBK TIME(0) nの場合は、

日、秒 --> MBK(n)

月、分 --> MBK(n + 1)

年、時間 --> MBK(n + 2)

この値は1秒ごとに自動更新します。停止させる場合は、S_MBK DATE(0) 0, S_MBK TIME(0) 0と0を指定します。

この場合指定アドレスは0となっていますが、MBK(0) ~ MBK(2)への書き込みは行われません。

MBK

書式

MBK(arg)

使い方

a=MBK(n)
a=MBK(n-Lng)
MBK(n)=a
b=MBK(n-Int)

機能

タッチパネルデータを参照、設定する。

解説

MBK配列は、タッチパネルと接続されると、メモリ共有される配列です。

MBK(n)はDTnに対応します。

a=MBK(n) タッチパネルデータをワード型で取り出す。

b=MBK(n-Int) タッチパネルデータを符号付ワード型で取り出す。たとえば値が&HFFF0の場合-16となる。

a=MBK(n-Lng) タッチパネルデータをロング型で取り出す。Hiワードがn+1アドレス

MBK(n)=式 タッチパネルデータにワード形で代入する。

MBK(n-Lng)=式 タッチパネルデータにロング形で代入する。

MBK(n)には以下の予約領域があります。

1) 文番号

MBK(7868) ~ MBK(7899) 実行中のプログラムの文番号です。ワード型です。

文番号が65535を超える場合は、

S_MBK LONG_PRG

を実行します。以後

MBK(7836)～MBK(7899)にロング型で文番号が格納されます。

2)バージョン番号

MBK(8053)には、バージョン番号が格納されています。

ファームウェアバージョンが1.12_60であれば、

pr MBK(8053) -> 11260

となります。

3)MBK(7900)～MBK(7999)までは、タッチパネル側でのRエリアとして扱われます。

Rエリアのバンク0～99がこのエリアに対応します。

MBK\$

書式

MBK\$(adr,val)

使い方

A\$=MBK\$(100,6)

機能

MBK配列を文字列として読み取る

解説

S_MBK a\$ adr cと対になる関数です。MBK配列上の文字列を読み出します。

MBK_CMD

書式

MBK_CMD

使い方

PRX MBK_CMD

機能

通信エラーキャラクタ

解説

MEWNET通信で処理できなかったコマンド。
PRX MBK_CMD で4142であれば、ABという意味です。

MBK_ERR

書式

MBK_ERR

使い方

PR MBK_ERR

機能

通信エラーカウンタ

解説

MEWNET通信のエラー回数を保持している変数です。

Int

書式

Int

使い方

IN(-1~Int)

機能

ワード型指定(符号付)

解説

S_MBK,MBK(),IN,OUT の符号付16bit読み取りを指定します。

```
10 S_MBK &H00008FFF 20~Wrd /* WORD write
20 PRINT MBK(20~Wrd) /* unsigned WORD read
30 PRINT MBK(20~Int) /* signed WORD read
40 OUT -1 -1~Wrd /* WORD write
50 PRINT IN(-1~Wrd) /* unsigned WORD read
60 PRINT IN(-1~Int) /* signed WORD read
RUN

36863 /* unsigned
-28673 /* signed
65535 /* unsigned
-1 /* signed
```

Lng

書式

Lng

使い方

MBK(20-Lng)

機能

ロング型(2ワード)指定

解説

S_MBK,MBK(),IN,OUT の値の32bitロング型での読み取り指定です。

```
10 S_MBK &H12345678 20-Lng /* LONG write MBK data area 20,21
20 PRX MBK(20-Lng) /* LONG read MBK data area 20,21
30 PRX MBK(21) MBK(20) /* WORD read
40 OUT &H87654321 -1-Lng /* LONG write memory I/O area -1~-4
50 PRX IN(-1-Lng) /* LONG read memory I/O area -1~-4
60 PRX IN(-4) IN(-3) IN(-2) IN(-1) /* BYTE read
RUN

12345678 /* LONG read
00001234 00005678 /* WORD read
87654321 /* LONG read
00000087 00000065 00000043 00000021 /* BYTE read
```


LONG_PRG

書式

LONG_PRG

使い方

S_MBK LONG_PRG

機能

プログラム番号のロング化

解説

タッチパネル用プログラム番号のロング化。

通常システムは、ワード領域MBK(7868)～MBK(7899)に、実行しているプログラムの文番号をセットします。プログラムが大きくなって番号が65535以上の値を持つ場合に。このコマンドでワード書き込みをします。この場合MBK(7836)～MBK(7899)に、プログラム文番号をロング整数書き込みします。

10 MEWNET 38400 1 /* RS-232C CH1 -> MBK-RS 38400bps
20 S_MBK LONG_PRG /* upper MBK(7836) -> long numeric

Wrd

書式

Wrd

使い方

IN(-1~Wrd)

機能

ワード型指定

解説

S_MBK,MBK(),IN,OUT の読み取りを16bit符号なしに指定します。

```
10 S_MBK &H00008FFF 20~Wrd /* WORD write
20 PRINT MBK(20~Wrd) /* unsigned WORD read
30 PRINT MBK(20~Int) /* signed WORD read
40 OUT -1 -1~Wrd /* WORD write
50 PRINT IN(-1~Wrd) /* unsigned WORD read
60 PRINT IN(-1~Int) /* signed WORD read
RUN

36863 /* unsigned
-28673 /* signed
65535 /* unsigned
-1 /* signed
```

文字列

Category

ADD_STR

書式

```
ADD_STR Str [Str]
```

使い方

```
ADD_STR "Win" a$  
ADD_STR "7"
```

機能

文字列のアpend

解説

ADD_STRは、指定文字列に文字列を追加します。

最初は、追加先の文字列変数の指定と、初期値を与えます。

```
ADD_STR "Win" a$
```

この時点で、a\$には、Winがコピーされます。

以後は追加文字列を指定するだけで、文字が追加されます。

```
ADD_STR "7"
```

この結果は、a\$がWin7となります。

ADD_STRは以下の記述によってヌル・コードも追加することができます。

```
ADD_STR chr$(0)
```

サンプルプログラムは、01,03,00,00,01,01を出力する場合の記述です。

```
CNFG# 2 "38400b8pns1NONE"  
CH=2  
ADD_STR CHR$(1) SEND$  
ADD_STR CHR$(3)  
ADD_STR CHR$(0)  
ADD_STR CHR$(0)  
ADD_STR CHR$(1)  
ADD_STR CHR$(1)
```

```
PRINT# CH STR_LEN|6 SEND$  
END
```

COMPOWAY

書式

```
COMPOWAY n m l str1$ str2$  
COMPOWAY str1$ v1 v2 v3 str2$
```

使い方

```
COMPOWAY 1 2 0 cmnd$ buff$  
COMPOWAY buff$ nod adr id rcv$
```

機能

OMRON COMPOWAYフォーマットの文字列生成および分解

解説

OMRON COMPOWAYは以下のようなコマンドフォーマットとなっています。

送信時: ADR+SADR+ID+CMND文字列

受信時: ADR+SADR+END+RES文字列

COMPOWAYコマンドは双方の文字列を効率よく生成、解析します。

生成:指定した、アドレス、サブアドレス、コマンド文字列(cmnd\$)をbuff\$に収納します。

```
COMPOWAY 1 2 0 cmnd$ buff$
```

分解:うけとった文字列buff\$を変数nod adr id にレスポンス文字列をres\$に収納します。

```
COMPOWAY buff$ nod adr id res$
```

nod,adr,idはCOMPOWAYの定型フォーマットに含まれる数値データです。

res\$は、コマンドによって応答が異なりますので、適宜、接続機器の仕様に基づいて、読み出し判定を行います。

なお、BCCエラーはinput# COMPOWAY コマンド実行後 rse_に反映されます。(0で正常、4はBCCエラーです)

```
*RS-485_SEND_READ  
_VAR data_len
```

```
cmnd_txt$=mrc_src$+hensu_shu$+str_adr$+bit_ichi$+yoso_su$+setteichi$
COMPOWAY node_no sub_adr sid cmnd_txt$ snd$
PRINT# 5 COMPOWAY snd$
INPUT# 5 COMPOWAY rcv$
IF rse_!=0 THEN
/* WHEN rse_ is 4 , BCC error happend , OTHER cases indicates RS-232c errors
PRINT "communication error"
END_IF
COMPOWAY rcv$ node_no sub_adr end_code res$
ptr_=res$+4
res_code=HEX(PTR$(4))
ptr_=res$+8
res_data$=PTR$(data_len)
RETURN
```

FORMAT

書式

FORMAT Strng

使い方

```
FORMAT " DatB=[s00.000]"  
FORMAT "D=S00000"
```

機能

STR\$()の展開様式を定義する。

解説

STR\$()は、FORMATコマンドで定義されていない限り、標準整数様式で文字列展開します。

STR\$(1234) -> " 1234" STR\$(-1234) -> "-1234"

FORMATコマンドでは15文字の範囲で出力フォーマットを決定できます。

```
FORMAT " DatA=[S 0.000]" --> DatA=[- 8.000]
```

```
FORMAT " DatB=[s00.000]" --> DatB=[+02.000]
```

フォーマットで指定した文字列のスペースが0のところに右つめで数値がはいります。

Sは符号で、ラージSでは、正の場合スペース、

スモールsでは、正の場合に+符号が与えられます。Sもsもつけないと、符号は付加されません。

```
FORMAT "0000年00月00日" /* 文字列書式設定  
DT$=HEX$(DATE(0)) /* 年月日文字列取得  
FORMAT "00時00分00秒" /* 文字列書式設定  
TM$=HEX$(TIME(0)) /* 時分秒文字列取得  
PR "(1)" DT$ TM$ /* 表示
```


*RUN結果

(1) 2007年11月07日 12時34分00秒

GET_VAL

書式

GET_VAL strg_val array(n) [FPn]

使い方

```
GET_VAL a$ a(1)
GET_VAL a$ a(1) 100
```

機能

文字列から数値を取り出し、配列に連続代入する。

解説

文字列に含まれる数値を配列に一括代入します。

引数は、文字列変数(XX\$)と配列変数array(n)に制限されています。

(mbk,x(n)には代入できません。)

三番目の引数を省略すると、小数点(ドット)もデリミタとされます。

三番目の引数に10,100,1000などの数値を設定すると、小数点を含む数字列を指定倍数の数値として配列に代入します。

```
10 DIM a(10)
20 a$="1111.12 -2222.13 3333.1 4444.5 345m-9730"
25 PRINT a$
30 FILL a(1) 99 777
50 GET_VAL a$ a(1)
60 PRA a(1)
65 PRINT "FP"
70 FILL a(0) 99 777
80 GET_VAL a$ a(1) 100
90 PRA a(1)
#run
```

```
1111.12 -2222.13 3333.1 4444.5 345m-9730
```

a(1)=1111
a(2)=12
a(3)=-2222
a(4)=13
a(5)=3333
a(6)=1
a(7)=4444
a(8)=5
a(9)=345
FP
a(1)=111112
a(2)=-222213
a(3)=333310
a(4)=444450
a(5)=345
a(6)=-9730
a(7)=777
a(8)=777
a(9)=777
#

POKE

書式

POKE arg1 arg2 .. (str\$+n)

使い方

POKE &H03 (a\$+0)
POKE &H41 42 (a\$+5)

機能

文字列データ変更

解説

文字列中のコードを指定コードに置き換えます。

指定コードは、NULL、その他のバイナリコードも入力できるため、CRCなどのバイナリデータの設定が簡単にできます。

引数は、8個まで設定することができ、最後の引数を文字列と文字位置の指定に用います。

文字列と文字位置指定は、(a\$+n)というように()で閉じます。

この場合は、a\$のn番目からという意味になります。

LIST

```
10 a$="1234567890"  
20 POKE &h0041 &h0042 &h0050 &h0051 (a$+3)  
30 PRINT a$  
#run
```

```
123ABPQ890  
#
```

PR_LCD

書式

PR_LCD string

使い方

PR_LCD DD\$
PR_LCD "ERR"

機能

文字列を表示

解説

与えられた文字列を7Segに表示します。表示可能な文字は、0～9,A～Zと一部の記号です。小文字や複雑な文字は表示できません。

(7Seg搭載機種のみ)

SERCH

書式

SERCH src\$ f\$

使い方

SERCH A\$ "C="

機能

文字列の検索。

解説

文字列を検索し結果はポインタ ptr_ に反映される。
以下の例では、PTR\$()と組み合わせて使用。

```
120 a$="adhjkashdjkas123_chuchu_tako_"
130 SERCH a$ "123"
140 c$=PTR$(8)
150 PRINT c$
#run

_chuchu_
#
```

STRCPY

書式

```
STRCPY src$ dst$ [m n]
```

使い方

```
STRCPY src$ dst$  
STRCPY src$ dst$ 6 3
```

機能

文字列の複写

解説

文字列をdst\$に複写。mはソース文字列(src\$)の複写開始位置(文字列先頭が0)。nは複写文字数。m,nを指定しなければ、全文字複写となります。

```
a$="012345abc"  
strcpy a$ c$ 6 3
```

であれば、c\$は"abc"となります。

```
a$="111111111011aaaaaa123baka_aabbanbQERaho_b11111229we48r9"  
PRINT a$  
PRINT LEN(a$)  
FOR i=0 TO 5  
  STRCPY a$ b$ i 10  
  PRINT b$  
NEXT i  
FOR i=20 TO 25  
  STRCPY a$ b$ i  
  PRINT b$  
NEXT i  
#run
```

1111111110
1111111101
1111111011
111111011a
11111011aa
1111011aaa
23baka_aabbanbQERaho_b11111229we48r9
3baka_aabbanbQERaho_b11111229we48r9
baka_aabbanbQERaho_b11111229we48r9
aka_aabbanbQERaho_b11111229we48r9
ka_aabbanbQERaho_b11111229we48r9
a_aabbanbQERaho_b11111229we48r9
#

SUBST

書式

SUBST str

使い方

```
b$="ABC123 &H1234FJ &HBCDEF1 "  
SERCH b$ "&H"  
ptr_=ptr_-2  
SUBST " $"
```

機能

文字列を置き換える。

解説

SUBSTは文字列ポインタの位置から与えられた文字列を上書きします。

```
70 b$="ABC123 &H1234FJ &HBCDEF1 "  
80 SERCH b$ "&H"  
120 ptr_=ptr_-2  
130 SUBST " $"  
140 ptr_=SERCH$("&H")  
150 ptr_=ptr_-2  
160 SUBST " $"  
170 PRINT b$  
#ABC123 $1234FJ $BCDEF1  
#
```

ASC

書式

ASC(str)
ASC(arg)

使い方

ASC(a\$)
ASC(4)

機能

文字列のアスキーコードを得る

解説

引数に文字列を与えると先頭の文字コードを返します。

0～4の数値を与えると、ptr_の位置から、与えられた数だけ文字コードを読み取っていきます。

このため、4文字以内の文字列比較を簡単に行うことができます。

```
10 a$="123abcABC456"  
20 PRINT ASC(a$)  
30 SERCH a$ "abc"  
40 FOR i=0 TO 4  
50 PRX ASC(i)  
60 NEXT i  
#run
```

```
49  
00000041  
00000041  
00004241  
00434241  
34434241  
#
```

CHR\$

書式

CHR\$(arg)

使い方

```
a$=CHR$(15)
print# 1 chr$(10)
```

機能

一文字生成

解説

a\$="slkd" などでは表現できない文字を生成します。
CHR\$(1) ==> SOH 等です。

DATE\$

書式

DATE\$(n)

使い方

a\$=DATE\$(1)+" "+TIME\$(1)

機能

日付文字列取得

解説

日付文字列を得ます。

DATE\$(0)-> 20090529

DATE\$(1)-> 5/29/2009

DATE\$(2)-> 5.29.2009

DATE\$(3)-> 2009-05-29

a\$=DATE\$(1)+" "+TIME\$(1)+": CNT="+STR\$(i)

HEX

書式

HEX(str)
HEX(arg)

使い方

```
b$="ABC123 &H1234FJ &HBCDEF1 "  
PRX HEX(b$)  
SERCH b$ "&H"  
PRX HEX(5)
```

機能

ヘキサ文字列の読取。

解説

文字列中のヘキサ文字列を読み取る。

通常はSERCHなどで場所を検索してから数値(桁数)を指定して読み取る。

SERCHを使用したあとは、文字列を使用しないで数値をいれる。

a\$="ABC123"のようにヘキサのみで成り立っている場合は、HEX(a\$)でも読み取る。

LIST

```
10 b$="ABC123 &H1234FJ &HBCDEF1 "  
20 PRX HEX(b$)  
30 SERCH b$ "ABC"  
40 PRX HEX(0)  
50 SERCH b$ "&H"  
60 PRX HEX(5)  
#run  
  
00ABC123  
00000123  
0001234F  
#
```

HEX\$

書式

HEX\$(arg)

使い方

```
a$=HEX$(100)
t$=HEX$(DATE(0))
```

機能

ヘキサデシマルで数値の文字列生成

t\$=HEX\$(DATE(0))では年月日を得ることができます。

解説

FORMAT指定がなければ、8文字のHEX-DECIMAL表現をします。
FORMAT指定があればそれに従います。FORMATの"S","s"は無効です。

```
10 FORMAT ""
20 PRINT HEX$(&H00ABCDEF)
40 FORMAT "&H0000"
50 PRINT HEX$(100000000)
#run
00ABCDEF
&HE100
#
```

LEN

書式

LEN(string)

使い方

```
print LEN(a$)
a=LEN(a$)
```

機能

文字列の文字数をカウントします。

解説

与えられた文字列の文字数を返します。

P\$

書式

P\$(val)

使い方

a\$=P\$(100)

機能

点データの文字列化

解説

点データエリアは、"SETP n strngs"コマンドによって、文字列配列のように使用することができます。
P\$()文字列として格納されたデータを取り出す関数です。

```
FORMAT "Test s "  
FOR i=1 TO 10  
a$="setp"+STR$(i-5)  
SETP i a$  
NEXT  
FOR i=1 TO 10  
PRINT P$(i)  
NEXT
```


PEEK

書式

PEEK(Str\$+n)

使い方

```
A=PEEK(b$+1)
B=PEEK(b$+LEN(b$)-1)
```

機能

文字列コードの取得

解説

PEEKは指定文字列の指定位置のコードを取得することができます。
通信用チェックサム計算などに役立ちます。

```
10 a$="123456789A"
20 PRX PEEK(a$)
30 PRX PEEK(a$+LEN(a$)-1)
#run

00000031
00000041
#
```

PTR\$

書式

PTR\$(m)

使い方

```
ptr_=a$  
ptr_=ptr_+10  
k$=PTR$(5)
```

機能

ポインタの位置からm文字

解説

ポインタの位置からm文字の文字列を取り出します。

文字列の中から必要とする文字列の切り出しが容易にできます。

ポインタ位置はptr_に反映されるため、この値を操作することによって文字列の切り出し位置を調整できます。

ptr_は、ptr_=a\$や、SERCHコマンドで初期化されます。

例1)場所・文字数が予め判明している場合の、PTR\$()の使用方法

例2)1つの文字列になっている2つの数値を、それぞれ独立した文字列に分解する方法です。

ポインタの位置の差を文字列の長さとして使用しているのに注意してください。

1)

```
FORMAT "" /* 文字列書式設定クリア  
TT$=HEX$(TIME(0)) /* 時分秒取得  
ptr_=TT$ /* 文字列の位置を取得  
ptr_=ptr_+2 /* ポインタ再設定  
HH$=PTR$(2) /* ポインタの位置から2文字切り出し  
ptr_=ptr_+2  
MM$=PTR$(2)
```

```
ptr_=ptr_+2
SS$=PTR$(2)
CL$=HH$+"."+MM$+"."+SS$ /* 文字列連結
PR "(1)" TT$ "->" CL$ /* TT$:元の文字列 CL$:合成後の文字列
```

```
#RUN
```

```
(1) 00123400 -> 12:34:00
```

```
#
```

```
2)
```

```
C41$="Mx+9.7042e+002 My+6.3210e+002"
```

```
' Serching the space position
```

```
a_=C41$
```

```
l_=LEN(C41$)
```

```
SERCH C41$ " "
```

```
b_=ptr_
```

```
'b_ is the space position.
```

```
a_=ptr_-a_
```

```
ptr_=C41$
```

```
C1$=PTR$(a_)
```

```
ptr_=b_
```

```
C2$=PTR$(l_-a_)
```

```
PRINT C1$
```

```
PRINT C2$
```

```
#RUN
```

```
Mx+9.7042e+002
```

```
My+6.3210e+002
```

```
#
```

SERCH\$

書式

SERCH\$(str)

使い方

```
ptr_=d$  
ptr_=ptr_+20  
a=SERCH$("we") : j$=PTR$(2)
```

機能

指定文字列を検索しポインタを検索後の位置に移動する。

解説

文字列を検索しポインタを検索後の位置に移動する。

SERCH\$()には検索すべき文字列の指定がありません。このため、ptr_をあらかじめ決定しておく必要があります。

```
ptr_=a$
```

```
a$="A=100 B=100"
```

```
ptr_=a$
```

```
ptr_=SERCH$("B=")-2
```

```
b$=PTR$(5)
```

b\$は、B=100となる。

実際には文字列の最初の検索は、文字列を指定できるSERCHコマンドを使用し継続SERCHに関数SERCH\$を用います。

サンプルプログラムでは、ptr_、PTR\$()と連携して文字列の切り出しを行っています。

```
10 a$="1234567890abcdefgABCDEFGH"
30 SERCH a$ "a"
35 s=ptr_-1 : e=SERCH$("A") : c=e-s-1
40 ptr_=s
50 c$=PTR$(c)
60 PRINT c$
#run
  abcdefg
#
```

STR\$

書式

STR\$(arg)

使い方

```
a$="data="+str$(A)
```

機能

数値の文字列化

解説

数値を数字文字列に変換します。

例えば以下のように実行すると、A\$は、"DATA= 1000"という文字列となります。

```
A=1000
```

```
A$="DATA="+STR$(A)
```

標準状態で、正の値には先頭にスペースを付加、負の値には "-" を付加します。

変換の様式は、FORMATコマンドで行います。

TIME\$

書式

TIME\$(n)

使い方

a\$=DATE\$(1)+" "+TIME\$(1)

機能

時間文字列取得

解説

時間文字列を得ます。

TIME\$(0)-> 00100957

TIME\$(1)-> 10:09:57

TIME\$(2)-> 10:09

a\$=DATE\$(1)+" "+TIME\$(1)+": CNT="+STR\$(i)

VAL

書式

VAL(str)
VAL(arg)

使い方

```
a$="a=1000 b=-1000 c=100"  
a=VAL(a$) : b=VAL(0) : c=VAL(0)  
a$="x=1000.123 y=-2120.1256 "  
SERCH a$ "x="  
PRINT VAL(1000)
```

機能

数字文字列の数値変換

解説

文字列中から数字文字列を探し出し、数値に変換します。

文字列中に複数の数字文字列が含まれている場合は、連続してVAL(0)を実行します。

このように、改めて文字列を指定せずargを0とした場合は、順々に数字列を取り出して数値に変換します。

argに、10～100000までの数値をいれると小数点の桁で指定倍します。

"X=123.4567"

というような場合は、VAL(10000)で読み取れば1234567という整数値が得ることができます。

```
10 a$="x=1000.123 y=-2120.1256 "  
20 PRINT VAL(a$)  
30 SERCH a$ "x="  
40 PRINT VAL(1000)  
50 ptr_=SERCH$("y=")  
60 PRINT VAL(10000)
```

文字列最初から、小数点を含む場合

```
10 A$="123.22 B=456.12 C=789.34"
```



```
80 ptr_=A$  
90 A=VAL(100)  
100 B=VAL(100)  
110 C=VAL(100)  
120 PRINT A B C
```

ptr_

書式

ptr_

使い方

ptr_=a\$

機能

文字列ポインタ

解説

タスク変数

文字列内の位置を示します。

```
10 a$=HEX$(DATE(0))
20 PRINT a$
30 ptr_=a$ /* set the pointer position
40 y$=PTR$(4) /* copy 4 characters
50 ptr_=ptr_+4 /* re-set the pointer position
60 m$=PTR$(2)
70 ptr_=ptr_+2
80 d$=PTR$(2)
90 PRINT y$ m$ d$
RUN

20081117 /* a$
2008 11 17 /* y$ m$ d$
```

VER\$

書式

VER\$

使い方

pr VER\$

機能

バージョンデータ取得

解説

バージョンデータが入っている文字列変数です。
バージョン番号は、MBK(8053)でも取得できます。

```
10 DIM a(10)
20 FILL a(0) 0
30 a$=VER$
40 PRINT "MPC_Version" a$
50 GET_VAL a$ a(0)
60 PRA a(0)
70 PRINT "MBK_8053=" MBK(8053)
#RUN
```

MPC_Version 1.11_29 2009/01/22

```
a(0)=1
a(1)=11
a(2)=29
a(3)=2009
a(4)=1
a(5)=22
a(6)=-2147483648
a(7)=-2147483648
a(8)=-2147483648
```

a(9)=-2147483648
MBK_8053= 11129
#

USB

Category

DIR

書式

DIR [n] [m]

使い方

DIR
DIR 100

機能

USBメモリのファイルリスト取得

解説

DIR とすると、USBメモリのディレクトリを表示します。

引数に番号を与えると、表示はしないで、以下のタイプのファイル名をMBKエリアに8文字+.???フォーマットでファイル名を複写します。(12文字なので6データごと)

**.P??

**.C??

**.T??

**.F??

MBK\$(n+4,12) ファイル名1

MBK\$(n+16,2) ファイル名2

デフォルトでファイル名50個まで、MBKエリアに保存されます。

それ以上必要な場合、あるいはそれ以下にしたい場合は、三番目の引数で、MBKエリアの上限値を指定します。

そして、以下の場所には取得データを保存します。

MBK(n)-->ファイル数

MBK(n+1)-->トータルファイル数

MBK(n+2)-->トータルディレクトリ数

MBK(n+3)-->USB使用容量 (Mbytes) (ただし ルートディレクトリの分のみ)

注) USBメモリの残容量の直接取得は、実際の空きブロックのカウントアップ処理が必要となり、2G以上のUSBメモリでは相当時間必要です。8Gの場合で1分程度となるため実用的ではありませんでした。代替方法として、ルートにあるファイルの総バイト数が判明しますので、総容量から消費バイト数を算出します。総容量は、USB(1,0)関数によってDIR命令実行後知ることができます。単位はMbyteです。

ON_USB,OFF_USB

書式

ON_USB
OFF_USB

使い方

ON_USB
OFF_USB

機能

USBポートのイネーブル・ディズエーブル

解説

メインCPUボードのUSBポートは、タスク29で、USBファイルアクセスシステムを起動することによって使用可能となります。ON_USBは必要な初期化とタスク29の起動を行います。

逆にOFF_USBは、ポートを停止し、タスク29も開放します。

RST_USB

書式

RST_USB

使い方

RST_USB : TIME 2000

機能

USBメモリプロセスの初期化

解説

USB動作エラー発生の場合(エラー56,68)は、RST_USB

コマンドでUSBメモリとUSBメモリプロセスを初期状態に戻し、再度、同じ処理を繰り返す。

```
ON_ERROR *ERROR_PROC
ON_USB
TIME 2000
FILE$=HEX$(DATE(0))+".CSV"
USB_DEL FILE$ /* Delete a file
DO
  FORMAT "00:00:00"
  WRITE_STR$=HEX$(TIME(0))+"/n"
  USB_WRITE WRITE_STR$ /* Write to the USB Memory
  TIME 5000
LOOP

*ERROR_PROC /* Error process
err_code=err_>>24 /* Get an error code
err_step=err_&&H00FFFFFF /* Get an error step number
PR "ERROR CODE" err_code "ERROR STEP" err_step
WAIT SW(192)==1 /* Restart
TIME 1000
RST_USB /* Reset the USB memory
TIME 2000
```

RESUME

USB_DEL {UDL}

書式

USB_DEL Str

使い方

USB_DEL "aaa.p2k"
UDL "aaa.f2k"

機能

USBメモリファイル抹消

解説

USBメモリ上の指定ファイルを抹消します。
ファイルが存在しない場合に実行してもエラーとはなりません。

USB_LOAD {UL}

書式

USB_LOAD strg

使い方

USB_LOAD "DEMO.F2K"
USB_LOAD "DEMO.F2K"

機能

USBメモリから、プログラムをロードする

解説

FTMWでUSBメモリ上にSAVEされた、プログラムデータをロードします。

USB_PLOAD {UPL}

書式

USB_PLOAD str

使い方

USB_PLOAD "PL3.P2K"
USB_PLOAD "PL3.P2K"

機能

点データをロード

解説

FTMWでセーブされたP2K,P68タイプのデータをUSBメモリからロードします。
上書きなので、新規データとする場合は、実行前にNEWPを実行します。
また、USB_PLOADでは、MBKデータにも対応します。

=データサンプル=

SETP 1 -200 9280 0 -12680
SETP 2 30880 9280 0 -13280
SETP 3 400 29400 0 -13280
SETP 4 31080 29280 0 -13680
SETP 101 8000 0 0 0
SETP 102 8000 8000 8000 4000
SETP 103 0 8000 0 0
SETP 104 0 0 0 4000
s_mbk 100 1
s_mbk 200 2
s_mbk 300 3
s_mbk 400 4
s_mbk 500 5
s_mbk 600 6
s_mbk 700 7
s_mbk 800 8

s_mbk 900 9
s_mbk 1000 10
s_mbk 30 7868

USB_PSAVE {UPS}

書式

USB_PSAVE P(n) cnt Str
USB_PSAVE MBK(n) cnt Str

使い方

USB_PSAVE P(1) 5000 "aa.p2k"
USB_PSAVE MBK(10) 1000 "aa.p2k"

機能

点データ、MBKデータの保存

解説

点データもしくはMBKデータのnからcnt個USBに保存します。

点データ保存で座標要素がすべて0の場合は、保存されません。

保存は、APPEND保存のため、ファイル既存の場合は、データ追加となります。

新規データとして保存する場合は、事前にUSB_DELコマンドでファイルを抹消します。

USB_DEL "AUTO.P2K"

USB_PSAVE P(1) 2000 "AUTO.P2K"

USB_PSAVE MBK(10) 1000 "AUTO.P2K"

この場合、AUTO.P2Kには、点データとMBKデータが保存されるため、リカバリデータとして使用することができます。

注意)電源オンオフをまたいだAPPENDはUSBメモリによってはできない場合があります。

USB_READ{URD}

書式

USB_READ String

使い方

USB_READ a\$
USB_READ -1

機能

USBメモリファイル一行読み込み

解説

USBメモリファイルの一行読み出しです。ファイル名はFILE\$で指定します。

順々に実行することによって、一行ずつ読み取ることができます。

ファイルの終端になると、関数EOF(0)が1を返すので、読み取りをそこで停止します。

無視して、URDを実行し続けると、またファイル先頭から読み始めます。

途中で読み出しを停止するには、USB_READ -1を実行します。

この処理によりファイルはクローズされます。

```
FILE$="AUTO.P2K"  
DO  
  USB_READ a$ : PRINT EOF(0) a$  
  IF EOF(0)==1 THEN : END : END_IF  
LOOP
```


USB_WRITE {UWR}

書式

USB_WRITE Strng

使い方

USB_WRITE "123.456"
USB_WRITE STR\$(n)

機能

USBメモリにアペンドライトする。(都度オープンクローズ)

解説

USBメモリに追記書込みする。

ファイル名の指定は予約文字列FILE\$(USB1 -> FILE1\$,USB2 ->FILE2\$)

引数の文字列には書き込む文字列をセットする。

都度、ファイルオープン・クローズするため、途中での電源断が発生しても、最終書き込みされたデータはUSBメモリ中に残る。

```
LIST
10 FOR k=1 TO 100
20 FORMAT "uwr_00.txt"
30 FILE$=STR$(k)
40 SEC=0
50 FOR SUM=1 TO 100
60 FORMAT "TEST_CNT=0000 /n"
70 A$=STR$(SUM)
80 FORMAT "APND_CNT=0000 /n"
90 B$=STR$(SUM+1000)
100 USB_WRITE A$+B$
110 NEXT
120 PRINT k SEC
130 NEXT
#
```

USB

書式

USB(0)

使い方

```
IF USB(0)!=1 THEN : GOTO *NOUSB : END_IF  
PR USB(1,0)-MBK(1000+3)
```

機能

USBメモリの有無

解説

USBメモリの有無を得ることができます。

USB(0)関数は、USBメモリが正しく装着されていると、1を返しますが、無いと、0を返します。

また、以下のように上位ワードに1をいれるとUSBメモリの総容量を返します。(Mbyte)

USB(1,0)

この値が有効になるのは、DIRコマンド直後か、電源オン時にUSBメモリが装着されている場合です。

```
FILE$="TEST"  
*RETRY  
DO  
IF USB(0)!=1 THEN : GOTO *NO_USB : END_IF  
USB_WRITE "TEST_WRITING /r /n"  
TIME 100  
DIR 1000  
IF USB(1,0)
```

CUnet

Category

CUNET

書式

CUNET arg1 arg2 arg3

使い方

CUNET 0 8 31
CUNET 8 8 15

機能

CUnetの初期化

解説

CUNET sa own end

saは領域確保の開始ブロック番号 0 ~ 63

ownは領域ブロック数。 1 ~ 32

endは全体で何ブロック共有するかという数 2 ~ 63

(ブロックはSA0 ~ SA63と表現されます。)

CUnetは8byteごとのブロックが64個あります。(512byte)

そのブロックの確保領域を定めてCUnetボードを初期化します。

初期化以後、CUnetのメモリエリアは2000番以上のIOアドレスとなります。

例えば、CUNET 0 1 32

とすると、SA0のみを確保します。これによりそのMPCでは、

OUT n SA0_B+m (m=0 ~ 7)でかきこめます。

ON/OFF は、ON SA0+m (m=0 ~ 63)でON/OFFできます。

他のステーションではIN/SWのみ有効で、同じ番号で読み取ります。

SA0,SA0_Bはそれぞれ予約定数で、ブロックごとに用意されています。

CUNET 8 8 32

とすれば、

ON/OFF は SA8より 8*8*8=512ビット制御します。

OUTでは、SA8_Bより8*8=64byte制御できます。

【通信レートの設定】

CUNETは通常12Mbpsで通信を行いますが、必要に応じて、6M,3Mbpsを選択することもできます。

この場合は、引数SAに&H80,&H40を論理加算します。

CUNET &H80|SA 6Mbpsに設定

CUNET &H40|SA 3Mbpsに設定

```
'display io
CUNET 0 8 31
DO
OUT IN(SA8_B) 2
OUT IN(SA8_B+1) 3
LOOP

'scan IO
CUNET 8 8 31
DO
FOR i=0 TO 15
ON SA8+i
WAIT SW(SA8+i)
TIME 5
OFF SA8+i
WAIT SW(SA8+i)==0
NEXT i
OUT 0 SA_B8 : OUT 0 SA_B8+1
LOOP
```

CU_POST

書式

CU_POST [n][VOID]

使い方

CU_POST
CU_POST 28

機能

CUnetメールサーバ

解説

CUnetメールサーバ起動コマンドです。

自動的に送られてきたCUnetメールを読み取り、転送命令にしたがってデータをP(n),MBK(n)に格納します。
また、要求にもとづいて(POST -n XXXコマンド)自己データを転送します。

CU_POSTコマンドは引数無しで実行すると、自動的に空きタスクを探しメールサーバを起動します。

引き当てられたタスク番号は、グローバル変数CUM_TASKに反映します。

また、引数を1～31の間で、与えると、その番号のタスクでメールサーバを起動します。

引数にVOIDを与えるか、指定タスク番号とVOIDをORして与えると、実行状況を表示します。

なお、メールサーバはCTRL_Aによって停止します。

メールサーバの動作状態は以下のグローバル変数で監視することができます。

CUM_ERR(エラーに)についてはOR更新、CUM_CNT(メールカウンタ)はインクリメント、
そのほかは、受信ごとに最新の値に更新されます。

CUM_TASK CU_POSTサーバの使用タスク番号

CUM_SRC 受信メールの相手アドレス

CUM_PNT 受信メールの種類 1: P(n) 2: MBK(n)

CUM_NUM 受信メールのP(n)もしくはMBK(n)のnの値

CUM_CNT 受信メールごとにインクリメント

CUM_ERR エラー各bitは以下のとおり。

BIT7: MAIL SEND ERROR

BIT6: 転送要求の応答が無い

BIT5: 通信停止

BIT4: 送信タイムアウト不正(通常0)

BIT3: 送信ブロック不正(通常0)

BIT2: 送信タイムアウト発生

BIT1: 送信相手不在

BIT0: 送信相手が受信待機となっていない

このコマンドを使用することにより、PCからMPC側の点データやMBKデータを書き換えたり、参照することができます。

これについては、USB-CUnetの資料を参照ください。

【参考資料】

メールの転送単位は、以下のとおりです。

P(n) 15個 Long型*4

MBK(n) 120個 Word型

メールのパケットは256byteで以下のような構成で最初の16byteをシステムエリアとして以下のように使い分けています。

256buffer= {Num(word)][Ary(byte)][Cmd(byte)][12byte reserved]P(n) ~ P(n+14)}

Numは、P(n)、MBK(n)、IN(n)の最初の場所nを示します。

Aryは P(n),Mbk(n),IN(n)の指定で、1でP(n),2でMbk(n)、3はIN(n)となります。

ただし、IN(n)の場合は1byteずつの扱いとなります。

33を指定するとバルク転送となり、一回の通信ですべての実I/O情報を得ることができます。(これはUSB-CUnetでのみ有効)

IN(n)ではnに負の値を指定するとメモリI/O領域を参照します。

Cmdは 引渡しが要求かの区別で、1で要求。2で引渡しです。

*Aryに33,Cmdに2を指定すると、I/Oの一括設定となりますので運用には注意が必要となります。

buffer = {

00 64 01 02 00 00 00 00 00 00 00 00 00 00 00 00

POST

書式

POST dst ary

使い方

POST 2 P(100)
POST 5 MBK(20)
POST -2 MBK(100)

機能

CUnet経由でのデータの転送

解説

CU_POSTが起動されている相手に対してデータを転送します。

POSTコマンド一回の転送単位は240byteです。

点データでは、15 point (240byte/16byte)

MBKデータでは、120個 (240byte/2byte)

【例】

POST 2 P(100)

SA=2のステーションに対して、P(100)～P(114)のデータが転送されます。

POST 3 MBK(20)

SA=3のステーションに対して、MBK(20)～MBK(139)のデータが転送されます。

また、dstを負の数にすると、データ転送を要求します。(注)SA0に対する要求は0のかわりに64を設定します。) この場合、自己側でもCU_POSTが起動されていなければなりません。応答が帰ってくるまでこのコマンドは待ち状態になります。

2秒以内に応答が無いとCUM_ERRのBIT6がセットされます。

【例】

POST -3 MBK(20)

SA=のステーションに対してMBK(20)～MBK(139)のデータを要求し、自分の同じエリアに書き込みます。

このコマンドにより、CUnetを備えたMPC間で、点データやMBKデータのデータの共有を行うことができます。しかし、応答速度は0.1秒～0.5秒です。リアルタイム性はありませんので、高速共有は、CUnetのメモリI/Oで行ってください。

送信が正常に完了したかどうかは、CUM_ERRを参照します。

送信エラーとなると、BIT7が1となり、詳細は、BIT0～BIT6に反映されます。

CUM_ERR

BIT7: MAIL SEND ERROR

BIT6: 転送要求の応答が無い

BIT5: 通信停止

BIT4: 送信タイムアウト不正(通常0)

BIT3: 送信ブロック不正(通常0)

BIT2: 送信タイムアウト発生

BIT1: 送信相手不在

BIT0: 送信相手が受信待機となっていない

POST 3 MBK(20)

IF CUM_ERR!=0 THEN : PRINT "X_ERR" CUM_ERR : END : END_IF

SA=3のステーションに対して、MBK(20)～MBK(139)のデータが転送され、正常に送信終了したかどうか確認します。

"POST

n"のように、データを指定しないと、相手側と自己のCU_POSTが起動されているかどうかの確認できます。

正常であれば、"Ok"と表示されます。

【サンプルプログラム】

MPC-AとMPC-BがCU-netで連結され、A側にのみタッチパネルが接続されたシステムを想定しています。

この時、MBK(1000)～をMPC-Aの操作画面、MBK(2000)～をMPC-Bの操作画面とします。

```
/*MPC-A
CUNET 2 2 32
MEWNET 38400 2
CU_POST
FORK 1 *SHARE_MBK
END
```

```
*SHARE_MBK
DO
POST 4 MBK(2000) /* MPC-B OUT AREA
POST -4 MBK(2200) /* MPC-B IN AREA
TIME 100
LOOP

/*MPC-B
CUNET 4 2 32
CU_POST
END
```

MKY

書式

MKY(val)

使い方

A=MKY(0)
PRX MKY(1)

機能

CUnet IC MKYの制御レジスタの読み取り

解説

CUnetチップであるMKY40の各レジスタの値を読み取ることができます。

MKY(0) SCR

MKY(1) BCR_SA(上位2bitは、Baud)

MKY(2) BCR_OA(上位2bitは、LFS,CP)

MKY(3) CHIP_CD: “ MKY4 ” を数値として返します。 prx MKY(3) -> 4D4B5934

MKY(4) MES(Mail Error Status)

MKY(5) SSR(System Status Register)

MKY(6) MFR(Member Flag Register 0-31

MKY(7) MFR(Member Flag Register 32-63

MKY(8) MCR(Member Care Counter)読み取りとクリア

MKY(9) LCR(Link Care Counter)読み取りとクリア

MKY(1)の上位2bitを除く値は、パワーオン直後、DSW1,DSW2の値となります。

これにより、CUnetのDSWの値によって、スタートアドレスを設定することができます。

MKY(3)により、ボードの有無を確認できます。

MKY(6),MKY(7)により、ネットワーク上のMKYの有無(電源ON,OFF)を確認することができます。

MKY(8),MKY(9)の値が頻繁に増加する(読み取るたびに0でない1以上の値がはいっている)ようであれば、通信の品質が外部要因で劣化しています。

RCV

書式

RCV(arg)

使い方

```
A=RCV(A$)
A=RCV(P(100))
A=RCV(DAT(10))
```

機能

メール受信

解説

RCV関数は、XMT関数と対で使用する、メール受信関数です。

CU_POST,POSTとは併用できません。

引数にP(n),X(n)~Z(n),MBK(n),配列、文字列を指定することができ、受信した256byteのデータを自動的に指定場所に格納します。

指定された時間以内(デフォルトは10秒)にメールが受信されないと、-3を返します。

タイムアウト時間の変更は、timer_に指定時間(0.1秒単位)を設定してから、RCV()を実行します。

-2は、CUM_ERRにエラー・コードがはいっている場合、-1は引数の指定が間違っている場合です。正常に受信すると、受け取ったメールの送信元の番号を返します。

```
--MPC A side--
LIST
10 CUNET 0 4 31
20 DIM a(100)
30 FILL a(0) 0
40 TIME 100
50 CUM_ERR=0
60 a$="1234567890"
70 IF XMT(8,a$)!=0 THEN : END : END_IF
```

```
80 IF RCV(a(1))!=8 THEN : END : END_IF
90 PRINT a(1) a(2) a(3) a(63) a(64)
#run

10 20 30 630 640
#
--MPC B side--
LIST
10 CUNET 8 4 31
20 DIM b(100)
30 TIME 100
40 CUM_ERR=0
50 IF RCV(b$)!=0 THEN : END : END_IF
60 PRINT b$
70 FOR i=1 TO 64 : b(i)=i*10 : NEXT
80 IF XMT(0,b(1))!=0 THEN : END : END_IF
90 END
#run

1234567890
#
```

SA

書式

SA(val)

使い方

ON SA(5)+0

機能

CUnetのSAに対応したON/OFF/SW番号を得る

解説

CUnetのステーションアドレスと、ON/OFF番号を関連づける関数。
SA5 の 最初のON/OFF番号は SA(5)となる。

SA_B

書式

SA_B(val)

使い方

OUT &H55 SA_B(5)

機能

CUnetのIN/OUTバンク番号を得る。

解説

CUnetのステーションアドレスと、IN/OUT命令のバンク番号を関連づける関数。
SA5の最初のバンクはSA(5)となる。

XMT

書式

XMT(dst,arg)

使い方

```
A=XMT(8,A$)
A=XMT(8,P(100))
A=XMT(16,DAT(10))
```

機能

メール送信

解説

XMT関数は、RCV関数と対で使用する、メール送信関数です。

CU_POST,POSTとは併用できません。

引数にP(n),X(n)～Z(n),MBK(n),配列、文字列を指定することができ、指定されたデータ256byteをdst宛に送信します。

正常に送信が終了すると、0を返します。

XMTを実行する前に相手のステーションでRCVが実行されている必要があります。

0以外の値がもどってきた場合は以下の原因による送信失敗です。

1:BIT0 送信先でRCVが実行されていない。

2:BIT1 送信先が無い

4:BIT2 メール送信通信不良

*サンプルプログラムはRCV()の項を参照してください。

SA0_B~SA63_B

書式

SA0_B-SA63_B

使い方

IN(SA0_B)

機能

CUnet SA番号

解説

Cunetのステーションアドレスに対応したI/Oバンク番号。

SA0~SA63

書式

SA0-SA63

使い方

ON SA0+5

機能

CUnet SA番号

解説

Cunetのステーションアドレスに対応したI/O番号。

浮動小数点

Category

AFFIN

書式

AFFIN n m k deg

使い方

AFFIN 2 1 3 i*10000

機能

点の回転演算

解説

P(n)をP(m)を中心にdeg度回転し結果をP(k)に代入します。
角度は10000倍した値を与えます。

サンプルでは、X方向の水平線を30度ccw方向に回転しています。

```
#setp 1 10000 20000 0 0
#setp 2 1010000 20000 0 0
#affin 2 1 3 300000
#pr p(3)
876025 520000 0 0
#
```

ATAN

書式

atan y r var [x]

使い方

atan 10000 1000 a
atan 100000 1000 a 173205

機能

ATAN演算

解説

ATAN浮動小数点演算を行います。結果は度で、整数化されます。
整数化する時の倍率をrによって決定することができます。

$\text{var} = r \times \text{atan}(y/x)$

注1) xを省略すると、xを10000とします。

注2) 結果(度)の範囲は-90 ~ +90となります。

例1) atan 10000 1 a

この計算は二等辺直角三角形のATAN値で、結果は45(度)です。

$a = 1 \times \text{atan}(10000/10000)$

例2) atan 17321 1000 a 10000

この計算は60度の直角三角形のATAN値です。

$a = 1000 \times \text{atan}(17321/10000)$

60度を1000倍するため、60000という値になります。

#atan 10000 1000 a
#pr a

```
45000
#atan 100000 1000 a 173205
#pr a
30000
#atan 17321 1000 a 10000
#pr a
60000
#
```

ATAN2

書式

atan2 y x var [r]

使い方

atan2 100000 100000 a3
atan2 100000 173205 a3 10000

機能

ATAN演算

解説

ATAN浮動小数点演算を行います。ATANとの違いは引数の順序のみです。

結果は度で、整数化されます。

整数化する時の倍率をrによって決定することができます。

$\text{var} = r \times \text{atan2}(y/x)$

注1) rを省略すると、rを10000とします。

注2) $y > x$ の場合は、 $\text{atan2}(x/y)$ を算出しその補角により角度を算出します。このため $x=0$ でも正しい値を返すことができます。

注3) 結果(度)の範囲は-90 ~ +90となります。

例1) atan2 10000 10000 a

この計算は二等辺直角三角形のATAN値で、結果は45(度)です。

rは省略されているので結果は10000倍されます。

$a = 10000 \times \text{atan}(10000/10000)$

例2) atan2 173205081 100000000 a 100000

この計算は60度の直角三角形のATAN値です。

$a = 100000 \times \text{atan}(173205081/100000000)$

60度を100000倍するため、6000000 という値になります。

```
#atan2 100000 100000 a
#pr a
450000
#atan2 100000 173205 a 10000
#pr a
300000
#
```


COS

書式

cos deg r var [sf]

使い方

```
cos 450000 100000 a
cos 4500000 100000 a 100000
```

機能

COS演算

解説

浮動小数点COS演算を行います。

$\text{var} = r \times \cos(\text{deg}/\text{sf})$

注) sfを省略すると、sfを10000とします。

例1) COS 600000 10000 a

```
#pr a
```

```
5000
```

```
#
```

cos(600000/10000)の演算でcos(60度)の意味です。

結果は0.5ですが、10000×0.5のため、5000となります。

```
#cos 450000 100000 a
#pr a
70711
#
#cos 4500000 100000 a 100000
#pr a
70711
#
```

FLOAT

書式

FLOAT equation1 equation2 ...

使い方

```
FLOAT A=1/3*10000  
FLOAT FP(1)=SIN(RAD(30))
```

機能

浮動小数点演算

解説

FLOATコマンド中の演算は、倍精度浮動小数点演算となります。

FLOATコマンド中整数変数への代入は、演算は倍精度でおこなわれ、代入時に整数化されます。

FLOATコマンドFP(n)への代入は、演算を倍精度でおこない、倍精度として代入します。

FLOATコマンド中、SIN,COS,TAN,ATAN,ACOS,SQR、RAD,DEG,VAL等の関数は、倍精度関数として使用されます。

```
' Get Pie  
FLOAT FP(6)=ACOS(SQR(3)/2)*6  
FLOAT FP(6)=(FP(6)-3)*10  
PRINT "PIE=3." FP(10000,6)  
' Get Napier  
a=1  
FLOAT FP(2)=1  
FOR i=1 TO 100  
a=a*i  
FLOAT FP(2)=FP(2)+1/a  
NEXT  
FLOAT FP(2)=(FP(2)-2)*10  
PRINT "Napier=2." FP(10000,2)  
,  
  
PRINT "Second order equation X*X+4*X+3"  
a=1 : b=4 : c=3
```

```

FLOAT FP(0)=(SQR(b*b-(4*a*c))-b)/2/a
FLOAT FP(1)=(SQR(b*b-(4*a*c))+1-b)/2/a
PRINT FP(10000,0) FP(10000,1)

```

GETDG

書式

GETDG n m deg

使い方

GETDG 1 3 deg

機能

角度計算

解説

ベクトルP(m)-P(n)のX軸に対する角度を算出します。
実際の計算は以下のように行われます。

$$\text{deg} = \text{ATAN}((Y(m)-Y(n))/(X(m)-X(n)))$$

角度degは10000倍された値で変数に返されます。

サンプルプログラムでは、

X(2)-X(1)=> 17320508 Y(2)-Y(1)=1000000

したがって、

ATAN(1000000/17320508)となり、30度となりますが、結果を10000倍するため、
300000が得られます。

```
#setp 1 10000 20000 0 0
#setp 2 17330508 10020000 0 0
#getdg 1 2 a
#pr a
300000
```

SIN

書式

sin deg r var [sf]

使い方

```
sin 450000 100000 a
sin 4500000 100000 a 100000
```

機能

sin関数演算

解説

浮動小数点SIN演算を行います。

$\text{var} = r \times \sin(\text{deg}/\text{sf})$

注) sfを省略すると、sfを10000とします。

以下はSINコマンド実行例です。

```
#SIN 300000 10000 a
#pr a
5000
#
```

この意味は、 $\sin(300000/10000)$ の演算で $\sin(30\text{度})$ の計算になります。
結果は0.5ですが、 $10000(\text{sf}) \times 0.5$ のため、5000となります。

```
#sin 450000 100000 a
#pr a
70711
#
```

sin 4500000 100000 a 100000
#pr a
70711

TAN

書式

tan deg r var [sf]

使い方

```
tan 300000 100000 a
tan 3000000 100000 a 100000
```

機能

TAN演算

解説

浮動小数点TAN演算を行います。

$\text{var} = r \times \tan(\text{deg}/\text{sf})$

注) sfを省略すると、sfを10000とします。

例1) TAN 450000 10000 a

```
#pr a
```

```
10000
```

```
#
```

tan(450000/10000)の演算でtan(45度)の意味です。

結果は1ですが、10000 × 1のため、10000となります

```
#tan 300000 100000 a
#pr a
57735
#
#tan 3000000 100000 a 100000
#pr a
57735
#
```

ACOS,ATAN

書式

ATAN(v)
ACOS(v)

使い方

FP(0)=DEG(ACOS(1/SQR(2)))
FP(1)=DEG(ATAN(1))

機能

逆三角関数

解説

ラジアン引数出力の倍精度逆三角関数です。浮動小数点演算式中でのみ、意味を持ちます。

FP(0)=DEG(ACOS(1/SQR(2)))
FP(1)=DEG(ATAN(1))

DEG

書式

DEG(v)

使い方

FLOAT A=DEG(ATAN(SQR(2)))

機能

度変換

解説

ラジアンを度に変換します。

```
#  
FLOAT A=DEG(ATAN(1))  
#pr A  
45  
#
```

RAD

書式

RAD(v)

使い方

FP(0)=SIN(RAD(45))

機能

ラジアン変換

解説

角度を度からラジアンに変換します。

を得るために、RAD(180)としても使用できます。

```
#FP(0)=RAD(180)
#FP(1)=TAN(RAD(45))
#pr FP(0) FP(1)
3.141593E+00 1.000000E+00
#
```

SIN,COS,TAN

書式

SIN(rad),COS(rad),TAN(rad)

使い方

FP(0)=SIN(FP(0))
FP(1)=TAN(RAD(30))

機能

三角関数

解説

ラジアン引数の倍精度三角関数です。FLOATコマンド中でのみ、意味を持ちます。

FLOAT FP(1)=SQR(SQ(SIN(RAD(i)))+SQ(COS(RAD(i))))

SQR

書式

SQR(v)

使い方

FP(3)=SQR(3)
A=SQR(3*3+4*4)

機能

平方根

解説

FLOAT コマンド中では、倍精度平方根取得関数です。整数演算中では、整数の開閉計算となります。

FP(0)=SQR(1+3+5+7)

VAL

書式

VAL(str)
VAL(0)

使い方

FP(0)=VAL(A\$)

機能

浮動小数点値取得

解説

FLOATコマンド中で、数字文字列を浮動小数点変数として取得

内部では、小数点以上、以下、またE指定の各数字列を倍精度整数として扱います。
このため、それぞれの数字列が倍精度整数(9桁以内)を超えるとエラーとなります。

```
A$="Mx+9.7042e+002 My+6.3210e+002"  
#FP(0)=VAL(A$)  
#FP(1)=VAL(0)  
#pr fp(0) fp(1)  
9.704200E+02 6.321000E+02  
#
```

FP

書式

FP(n)
FP(m,n)

使い方

FP(0)=1000
STR(FP(100,1))

機能

浮動小数点変数配列

解説

FP(0) ~ FP(15)まであり、FLOATコマンド中で倍精度浮動小数点変数として使用できます。

FLOAT FP(1)=1000

この場合、FP(1)には浮動小数点倍精度型として1000が保存されます。

また、VALと組み合わせることによって、指数表現のデータを格納することができます。

```
a$="Mx+9.7042e+002 "
```

FP(2)=val(a\$)とすれば、FP(2)は、9.704200E+02 としてデータが格納されます。

FP(n)を整数化して使用する場合は、FP(1,n)とします。

倍率が必要な場合には、1のかわりに1~10000までの値をいれると、指定倍数ののち整数化されます。

なお、内容の確認には、print文が使用できます。

```
#a$="Mx+9.7042e+002 My+6.3210e+002"  
#fp(2)=val(a$) fp(3)=val(0)  
#pr fp(2) fp(3)  
9.704200E+02 6.321000E+02  
#
```

AD_DA

Category

DA

書式

DA val [ch]

使い方

DA 1000 1
DA 2000

機能

DA出力

解説

MPC-AD12のDA出力を設定します。0～4095の値を指定します。標準設定で1mV/1digit。
設定された値は2mSEC以内にDA出力に反映されます。

MPC-AD12のDA出力は4CHあり、0～3までを指定できます。

さらに1枚MPC-AD12を追加した場合は、DA出力CH番号として4～7が割り当てられます。
2番目の引数でCHを指定しますが、省略するとCH0となります。

GET_AD

書式

GET_AD CH ARRAY() Cnt [ms]

使い方

```
GET_AD 0 X(1090) 360 4
GET_AD 1 Z(1090) 100
GET_AD 0 X(100) (ch,100)
```

機能

ADデータの連続取得

解説

CPU側の1msecタイマーを利用したMPC-AD12からのリアルタイムデータ取得です。
唯一の1msecタイマーで起動する機能のため、マルチタスク的な使い方はできません。

コマンドでは、取得チャンネル、データの収納配列、獲得数、msec単位での取得インターバルを指定します。
インターバルを省略すると1msec間隔となります。

このコマンドは、データの取得完了を待たずに終了します。

終了監視は、関数GET_AD(0)によって行います。この関数が1を返している間は、データ取得中です。

サンプルは、MPC-1000でステップモータを回しながら4msec間隔で、ADデータを取得するものです。

配列には点データ、MBK配列、DIM宣言配列を使用することができます。

この例では、データの更新確認を、X(1449)の値の更新によって行っています。

なお、データ配列にX()を指定しCntの上位ワードに2～4の値を設定すると、同時に複数チャンネルデータ取得します。

```
GET_AD 0 X(100) (3,100)
```

この場合、X(n)

SET_AD

書式

SET_AD [args]

使い方

```
SET_AD 10 10 10
SET_AD AD1 10 10 10
SET_AD AD7890_10 30 30 30 30
SET_AD AD1 AD7890_10
```

機能

AD設定

解説

SET_ADコマンドは、
ADコンバータのタイプや平均値サンプリングのサンプル個数を指定します。
AD(1,ch)での読み取りはデフォルトで1msec*8個での平均値となっていますが、
このサンプル個数を2～127の範囲で指定することができます。

たとえば、以下のコマンドラインでは、

```
SET_AD 10 10 10 10 20 20 20 30
```

ここでは

```
CH0 10,CH1 10,CH2 10,CH3 10 ,CH4 20,CH5 20, CH6 20,CH7 30
```

と平均値個数を設定しています。

2枚目のMPC-AD12に対しての設定は以下のように行います。

```
SET_AD AD1 10 10 10 10 20 20 20 30
```

ADコンバータをAD7890-10に交換した場合には以下を実行します。

```
SET_AD AD7890_10
```

これは、AD7890-10が負の電圧に対して2048～4095の値を割り当てることに対する補正。

2枚目のMPC-AD12をAD7890-10に交換した場合は、以下のとおりです。

SET_AD AD1 AD7890_10

なお、AD1 AD7890_10は予約定数で、-10の値を持ちます。

```
FORK 1 *disp
  END
*disp
dd=0
M=400
SET_AD AD7890_10
SET_AD 40
FORMAT "S000"
DO
t=AD(1,0) : t2=AD(1,2)
d=(M-t)*2 : IF d>35 THEN : d=25 : END_IF
IF d
```

AD

書式

AD(ch)
AD(fnc,ch)

使い方

```
A=AD(0)  
IF AD(1,7)>500 THEN
```

機能

MPC-AD12のデータ取得

解説

MPC-1000/MPC-N816/MPC-1200のAD機能は、AD(20)～を使用します。これについては、単純な読み取り変換機能だけで、MPC-AD12のようなマクロ機能はありません。

以下は、MPC-AD12に対する説明です。

【1msecサンプリング】

関数AD(ch)はADコンバータの変換値を返します。ch番号は、0～7です。

このデータは、1msecごとに更新されています。

MPC-AD12をさらに一枚追加した場合は、ch番号として8～15を指定します。

返される値は、AD7890-4(標準出荷状態)搭載で0～4095 1mV/1digit

AD7890-10搭載の場合は、-2048～20471mV/1digit

です。

平均値を得る方法

AD(1,ch) 平均値を返します。平均をとるデータ数の指定はSET_ADコマンドで指定します。

(デフォルトは8個ごとの平均値)

【自動連続データ取得】

MPC-AD12は1msecごとにデータを取得しており、832連続してその値を取得、参照できます。

AD(2,ch) スタート連続データ取得 1msecサンプリング
AD(4,ch) スタート連続データ取得 2msecサンプリング
AD(3,ch) 取得完了待ち

データの取り出しは、AD_D(0,n) n:0 ~ 831

【8CH 自動連続データ取得】

chを8に指定すると全ch同時サンプリングします。この場合レートは1msec固定。

各ch 104個のデータを取得します。(0.1秒)

AD(2,8) スタート連続データ取得 1msecサンプリング

AD(3,0) 取得完了待ち

データの取り出しは、AD_D(ch,n) ch:0 ~ 7 n:0 ~ 103

【その他の機能】

MPC-AD12には、30 μ sec ~ 100 μ secの高速サンプリングも用意されています。

これについては、ハードリファレンスを参照ください。

```
'1msec SAMPLING
SYCSCLK=0
FOR i=0 TO 1440 STEP 2
WAIT i==SYSCLK
X(i+1000)=AD(0)
NEXT
```

```
'Bulk Sampling
SYSCLK=0
dmy=AD(2,ch)
PRINT AD(3,ch) SYSCLK "1msec"
PRINT "dump"
FOR i=0 TO 800 STEP 50
PRINT i AD_D(0,i)
NEXT
```

```
'8CH Bulk sampling
dmy=AD(2,8)
PRINT AD(3,0) SYSCLK
FOR i=0 TO 100 STEP 10
PRINT i AD_D(0,i) AD_D(1,i) AD_D(2,i) AD_D(3,i)
NEXT
```

AD_D

書式

AD_D(ch,n)

使い方

a=AD_D(0,1)

機能

連続取り込みデータの読み出し

解説

連続サンプリングのデータの取り出し。

単chの場合は AD_D(0,n) n: 0 ~ 831

全chの場合は、AD_D(ch,n) ch:0 ~ 7 n:0 ~ 103